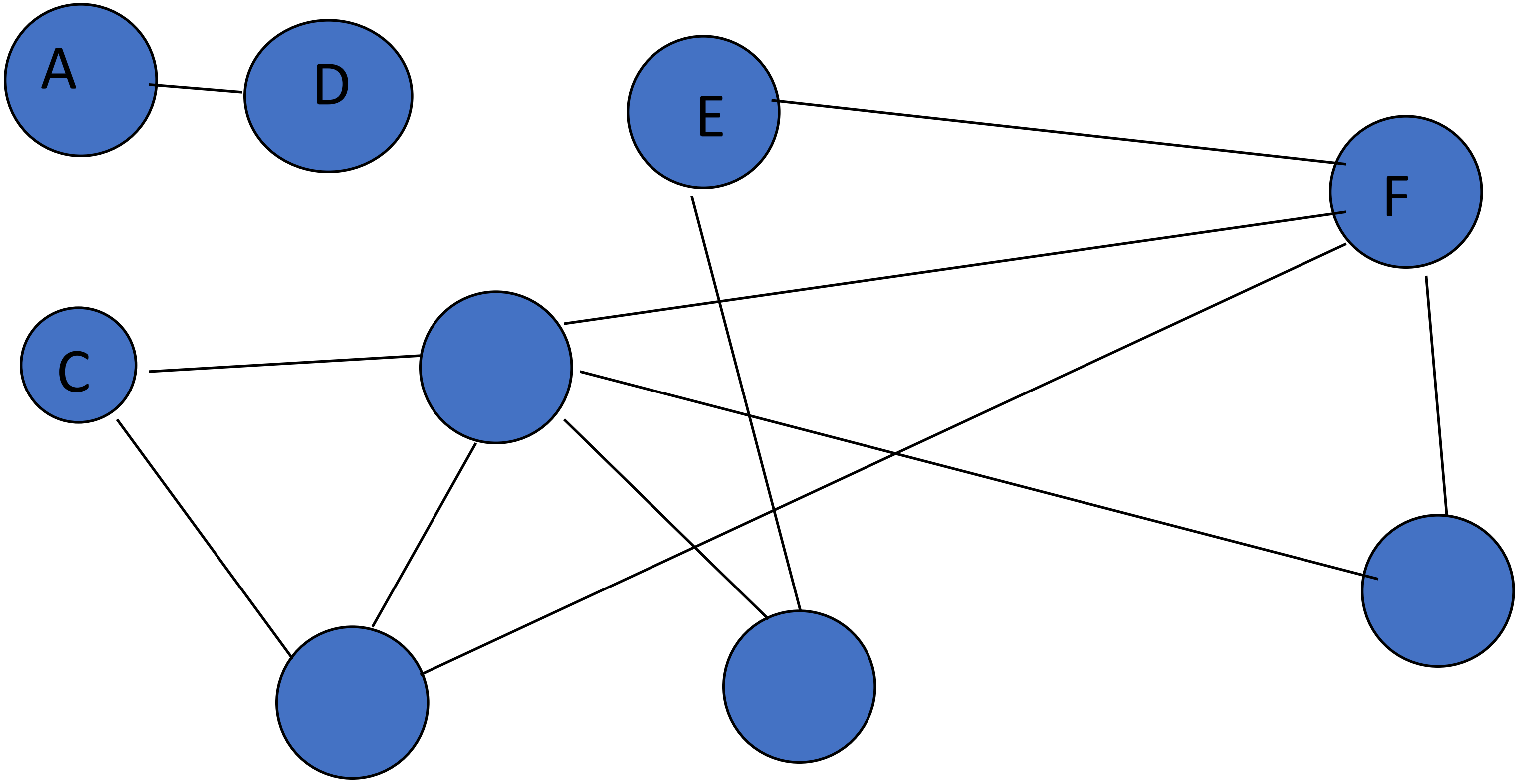
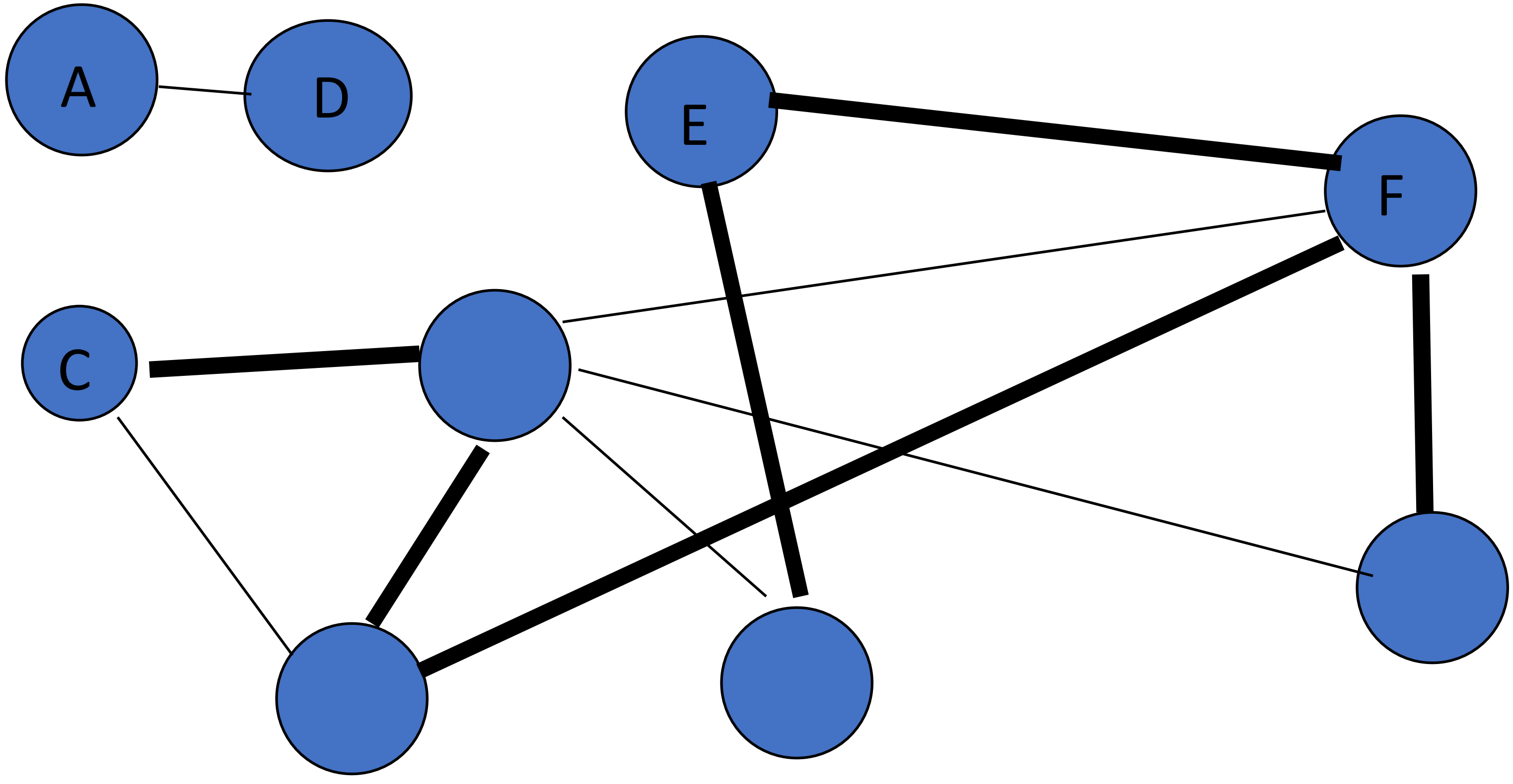
Dynamic Connectivity

Valerie King
University of Victoria
BC Canada

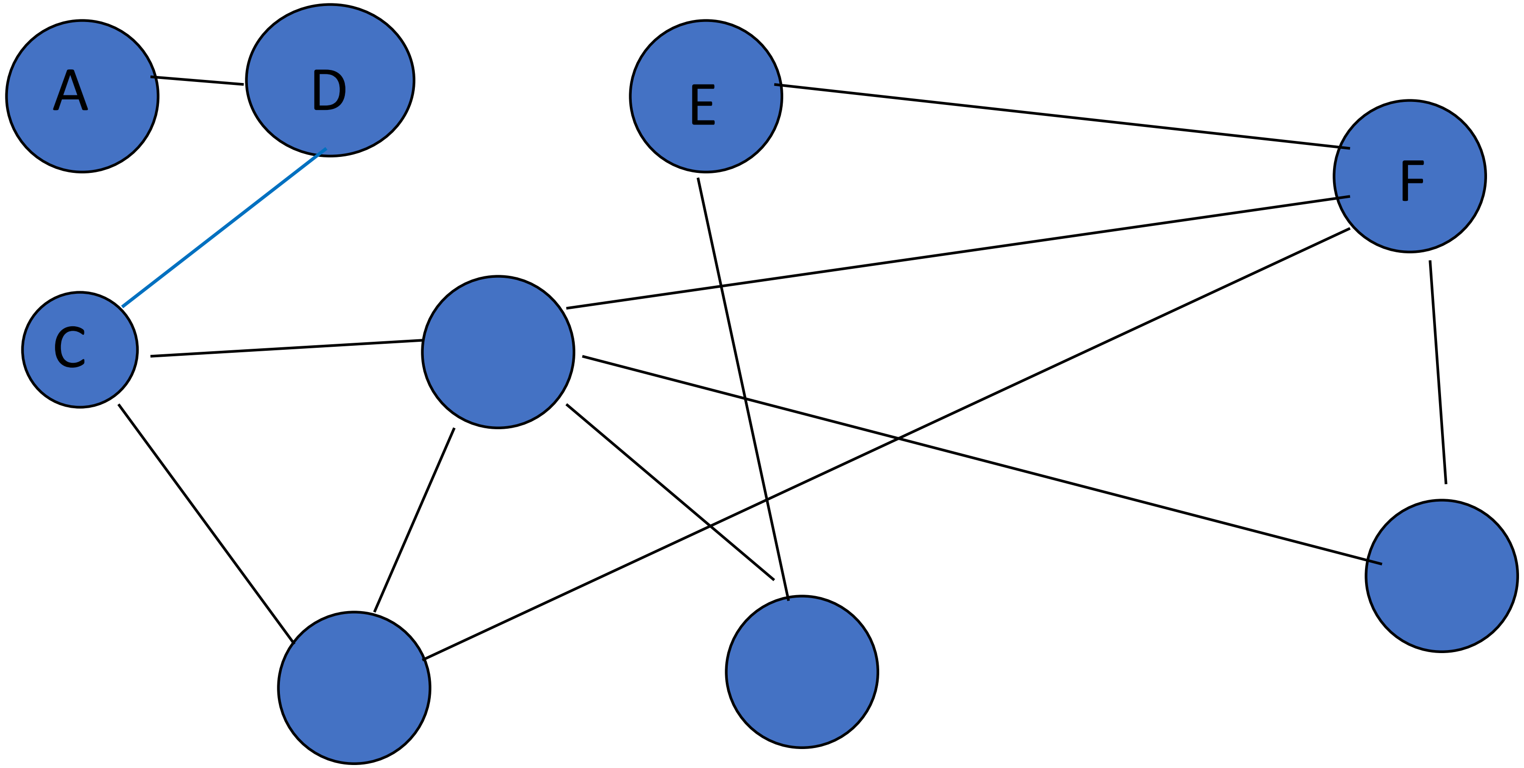
Determining connectivity in a graph is easy!



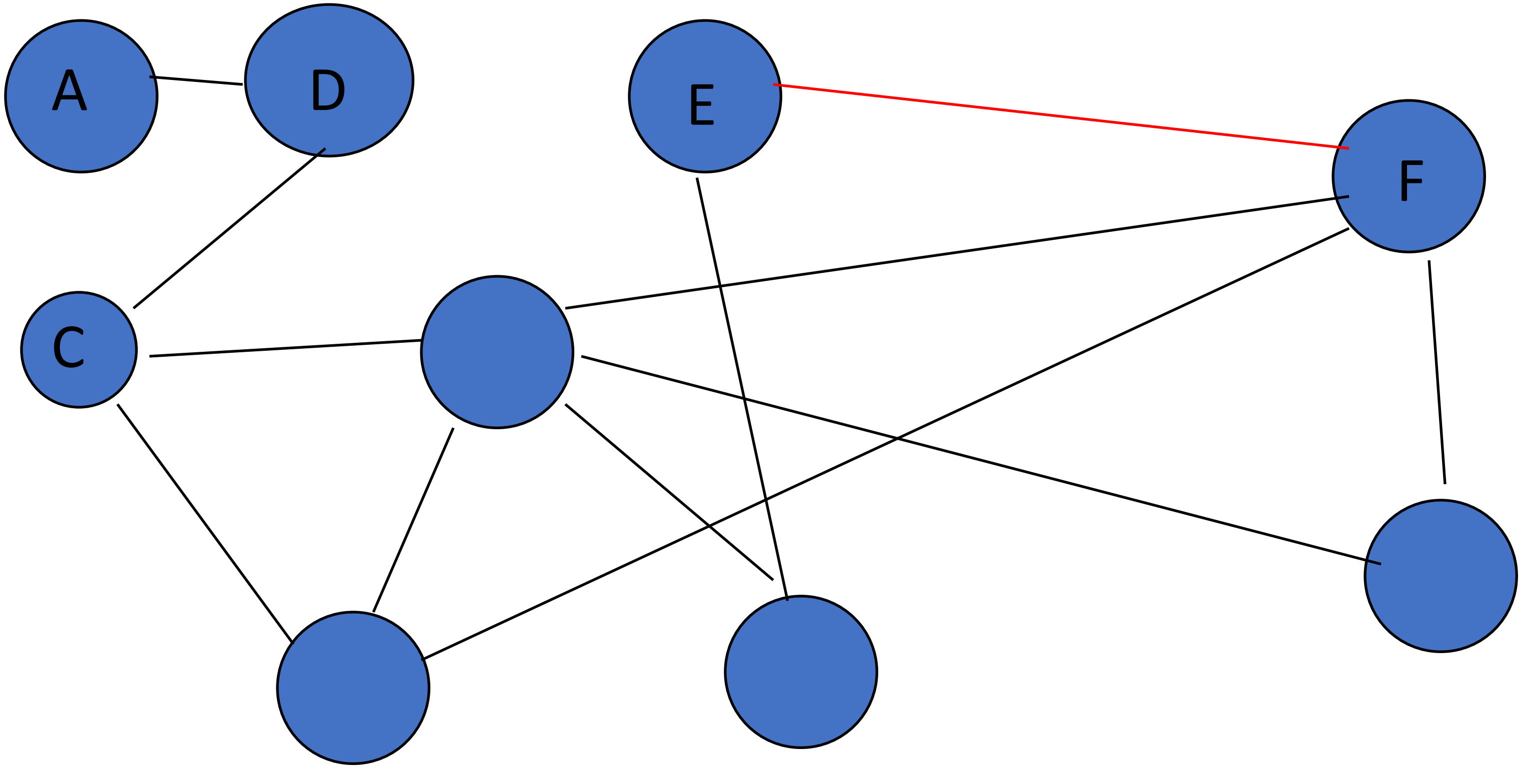
Determining connectivity in a graph is easy!



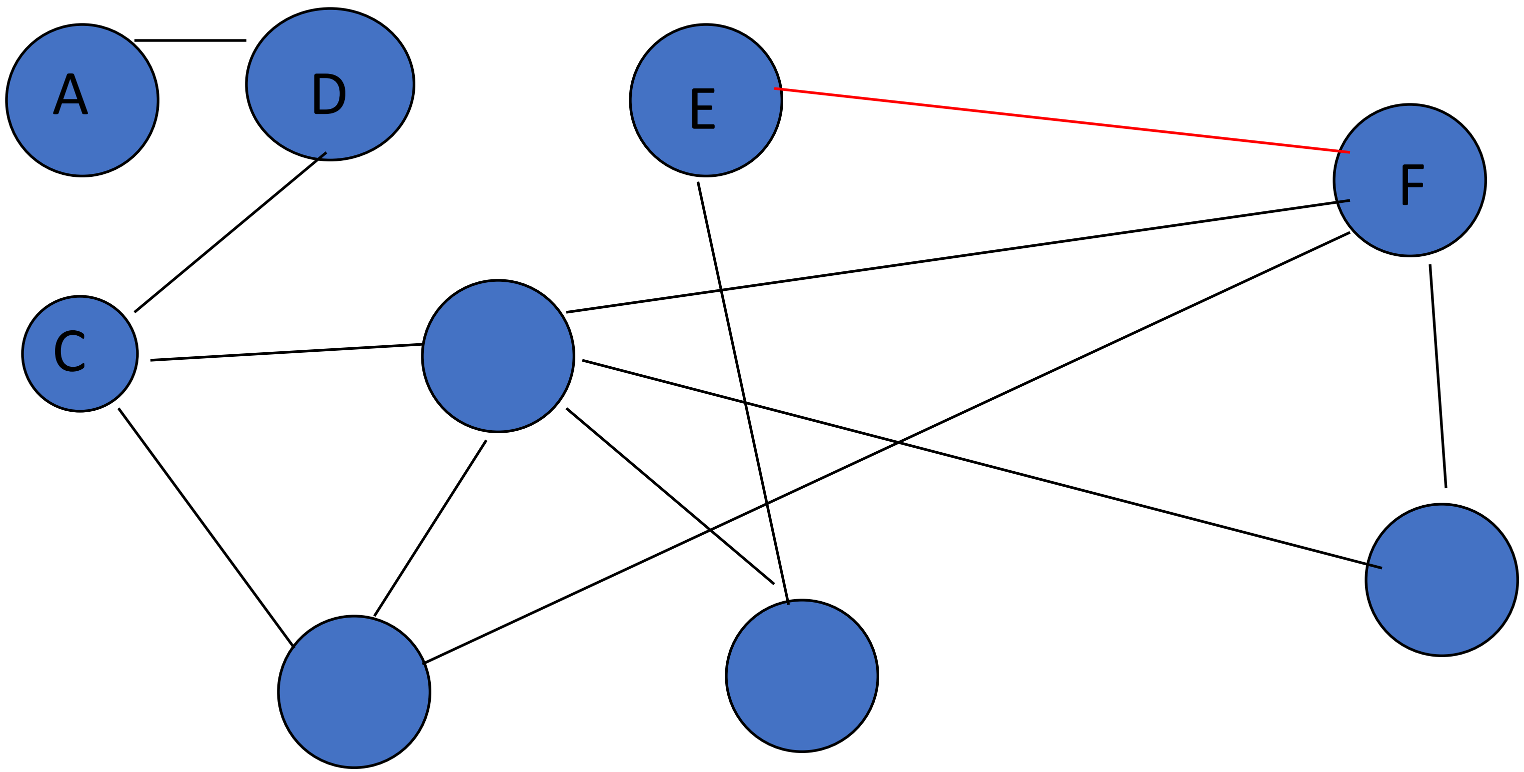
Update: Insert edge {C,D}



Update: Delete edge {E,F}



QUERY(D,F): Are D and F?



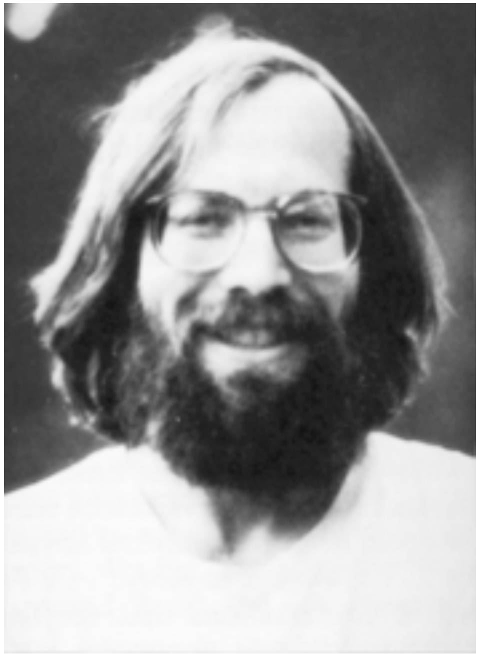
Challenge:

n=number of nodes, m=number of edges

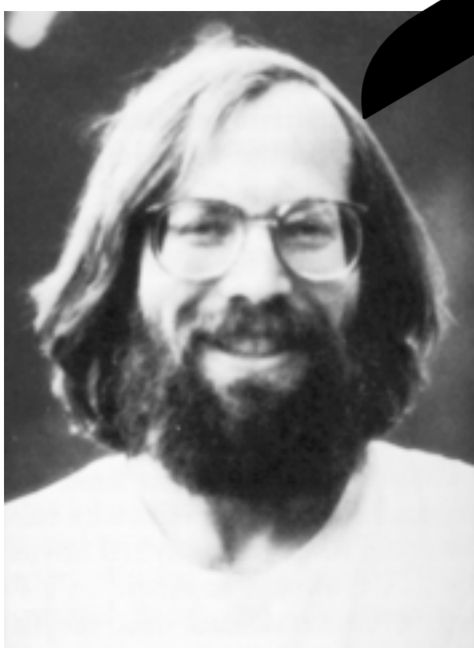
How to avoid O(m) cost of recomputing spanning forest with each update or running O(m) search for each query?

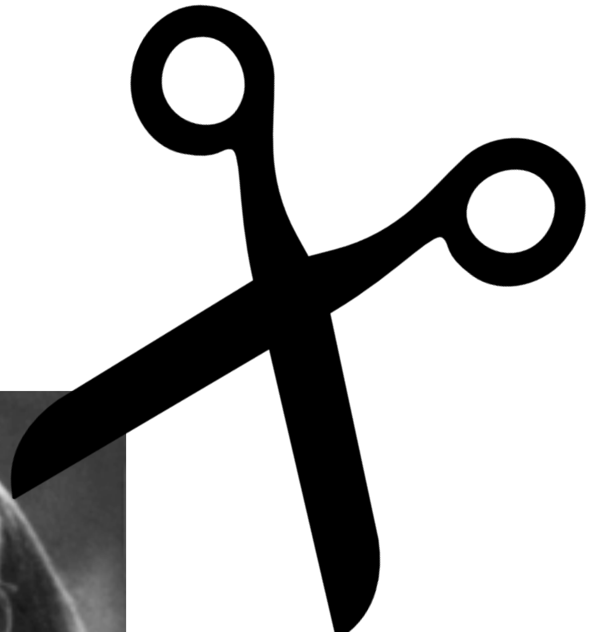
1960's and 70's

- Edge insertions only
- Union-Find data structure and
- Tarjan's $\alpha(m,n)$ amortized analysis



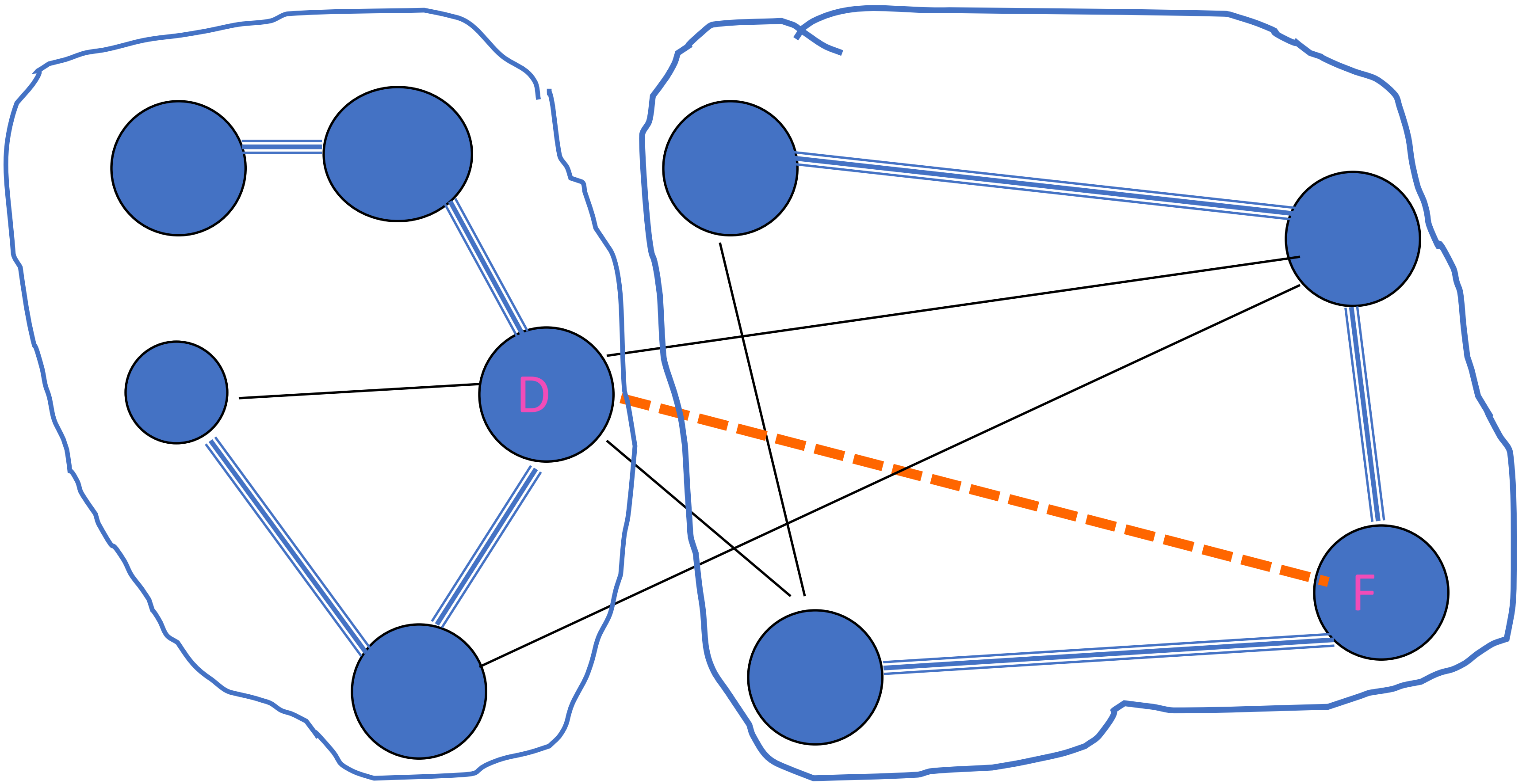
Deletions are much harder





Techniques rely on
maintaining a
spanning forest

When a tree edge is deleted…



How can we find a replacement edge?

A brief* history

Partially dynamic

- 1960's: Union-find insertions only (amortized) Tarjan's analysis (1975)
- 1981: Deletions-only (amortized) O(mn) Even-Shiloach ;
  - improved to O(m + n polylog) (Monte Carlo) Aamand et al (2023)

Fully Dynamic

- 1983 O(√m) topology trees Fredrickson
- 1992,7: O(√n) sparsification Eppstein, Galil, Italiano, Nissenzweig
- 1995,8: $O(\log^2 n)$ amortized
  - (Las Vegas) Henzinger,K (1995) as improved by Henzinger, Thorup (1997)
  - (deterministic) Holm, de Lichtenberg, Thorup [HDT] (1998), improved by Thorup; Huang, et al. to $O(\log n (\log \log n)^2)$ (higher query time) (2022)
- 2013: polylog worst case Monte Carlo Kapron, K, Mountjoy [KKM];
  - improved by Gibb, et al; Wang (2015).
- 2017: $n^{o(1)}$ worst case Las Vegas Nanongkai, Saranurak, Wullff-Nilson
- 2020: $n^{o(1)}$ worst case deterministic Chuzhoy, et al

In a variety of models

- Sequential
- Streaming
- Distributed
  - CONGEST, local, MPC
  - Synchronous/Asynchronous
- Parallel and Batch Parallel

Leading to related questions...

- Dynamic minimum spanning tree
- Dynamic tree data structures

  ET trees (1995) Henzinger, K
- Shortest paths, transitive closure (directed, weighted, all pairs and single source)
- Lower bounds in the cell probe model, streaming and distributed, using communication and information theory, conditional lower bounds
- Maintaining expander graph decompositions
- Distributed broadcast with sublinear communication

Talk Outline

- Review of some important ideas
- ET Tree and batch parallel implementation of the Monte Carlo [KKM+] method
- Application of the XOR method to distributed networks

A Simple problem , but lots of interesting ideas….

1. Union find
2. Even-Shiloach-Smaller components, breathfirst search tree
3. Hierarchical clustering (topology trees)
4. Sparsification
5. Randomized dynamic decomposition
6. Deterministic dynamic decomposition
7. ET Trees
8. XOR Method

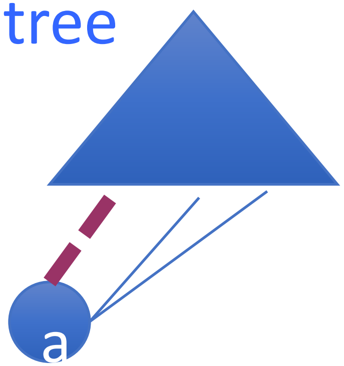
Idea 1: Union find—disjoint trees

- Create a node x for every node in graph and maintain a tree for each connected component
- find(x) returns root of tree containing x
- query (x,y): find(x)=find(y) iff x and y are connected
- Insert (x,y): if find(x)≠find(y), union(x,y)
- union(x,y): union by weight-- find(x) becomes child of find(y) if tree containing x is smaller (union by weight)
- While going up to root, set all pointers in tree to root (path compression).

IDEA 2: Even Shiloach deletions only

In Parallel: To delete {a,b}

A. Maintain breadth-first search tree



.

Pick next "hook" to higher level or drop a level and spend

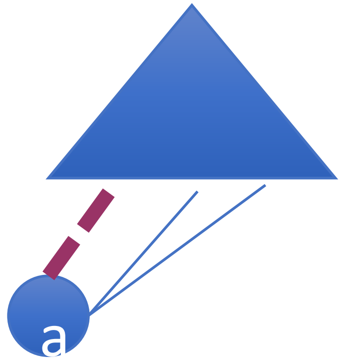
O(deg) to reset list of hooks

For total cost O(m*depth)

IDEA 2: Even Shiloach deletions only

In Parallel: To delete {a,b}

A. Maintain breadth-first tree



.

Pick next "hook" to higher level or drop a level and spend

O(deg) to reset list of hooks

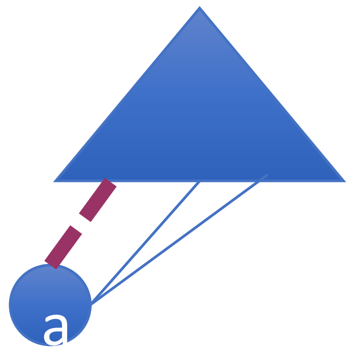
For total cost O(m*depth)

Led to dynamic shortest path algs

IDEA 2: Even Shiloach deletions only

In Parallel: To delete {a,b}

A. Maintain breadthfirst tree



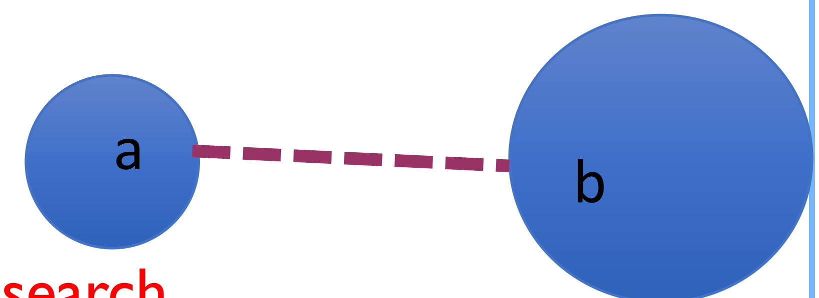
.

Pick next hook on higher level or drop a level and spend

O(deg) to reset hooks

Total cost-O(m* depth)

B. See if deletion splits graph into two components



search

Until first search ends

search

cost~ # of edges in smaller

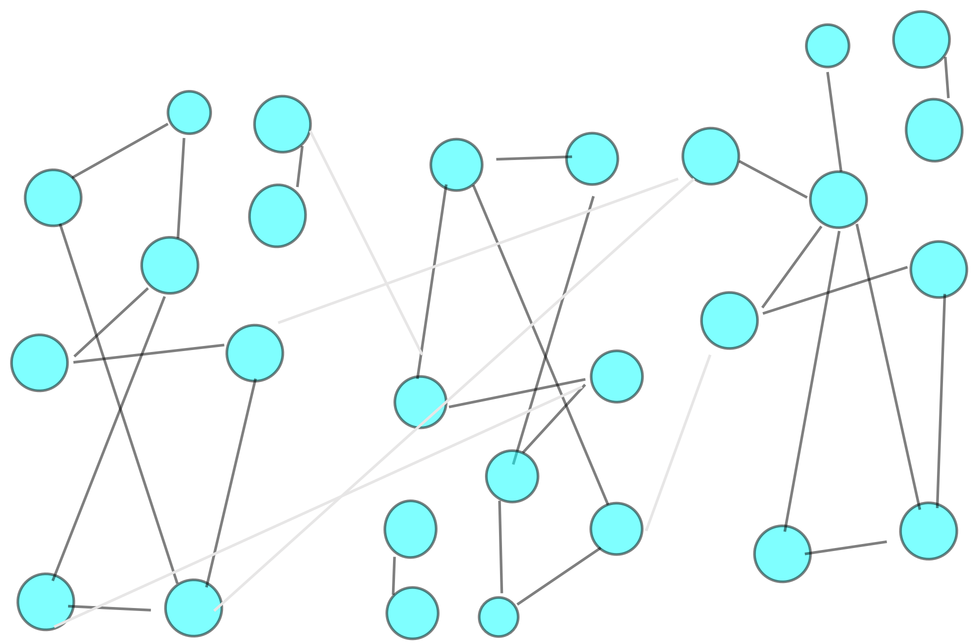
Each edge appears in ≤ lg m smaller components

Used for dynamic shortest path

Used for dynamic decomposition

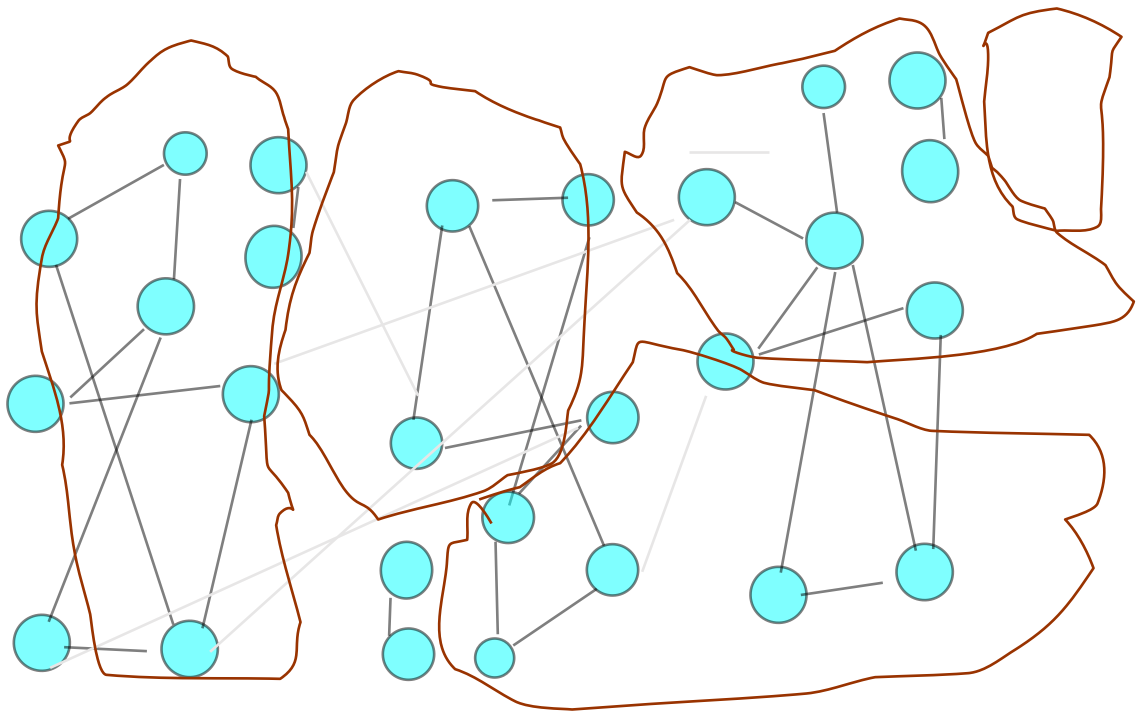
Idea 3. Hierarchical clustering (via topology trees)

Change to degree 3 graph by adding nodes



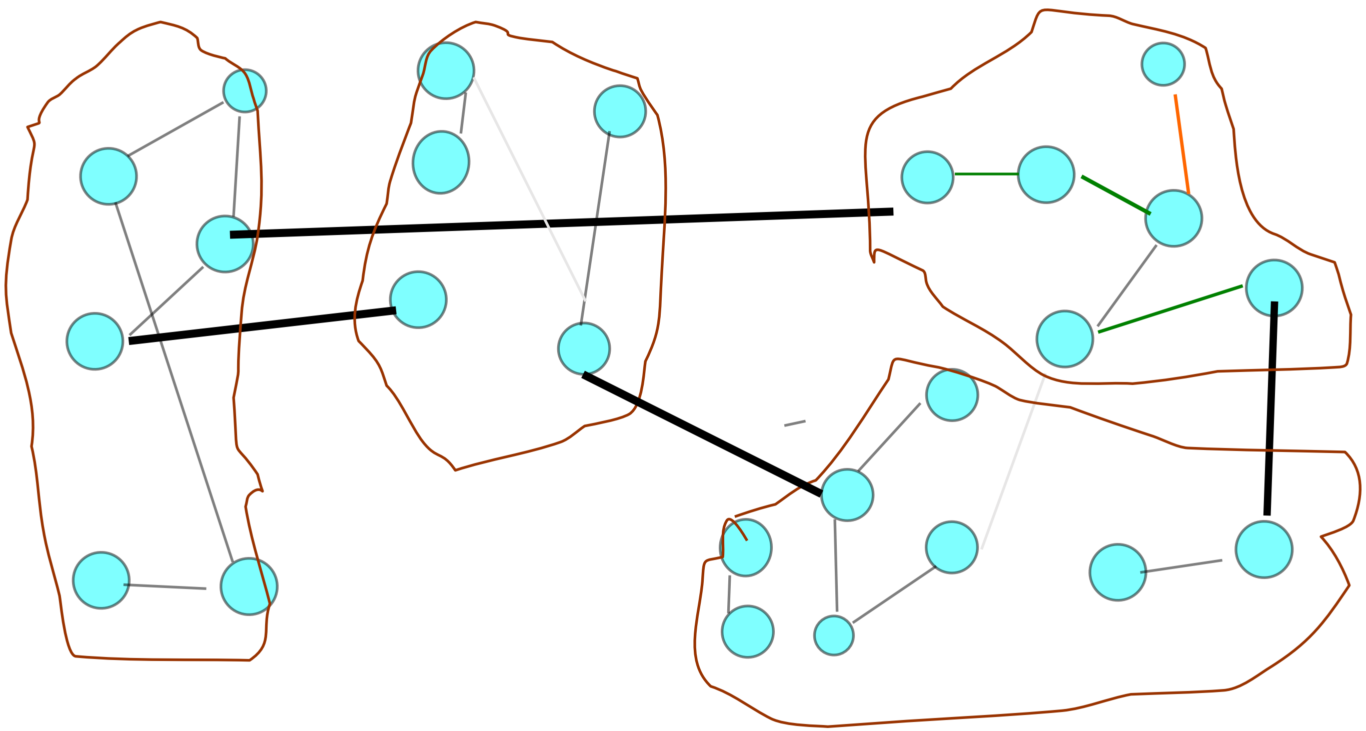
3. Hierarchical Clustering

Form m $^{1/3}$ connected clusters of size m$^{2/3}$



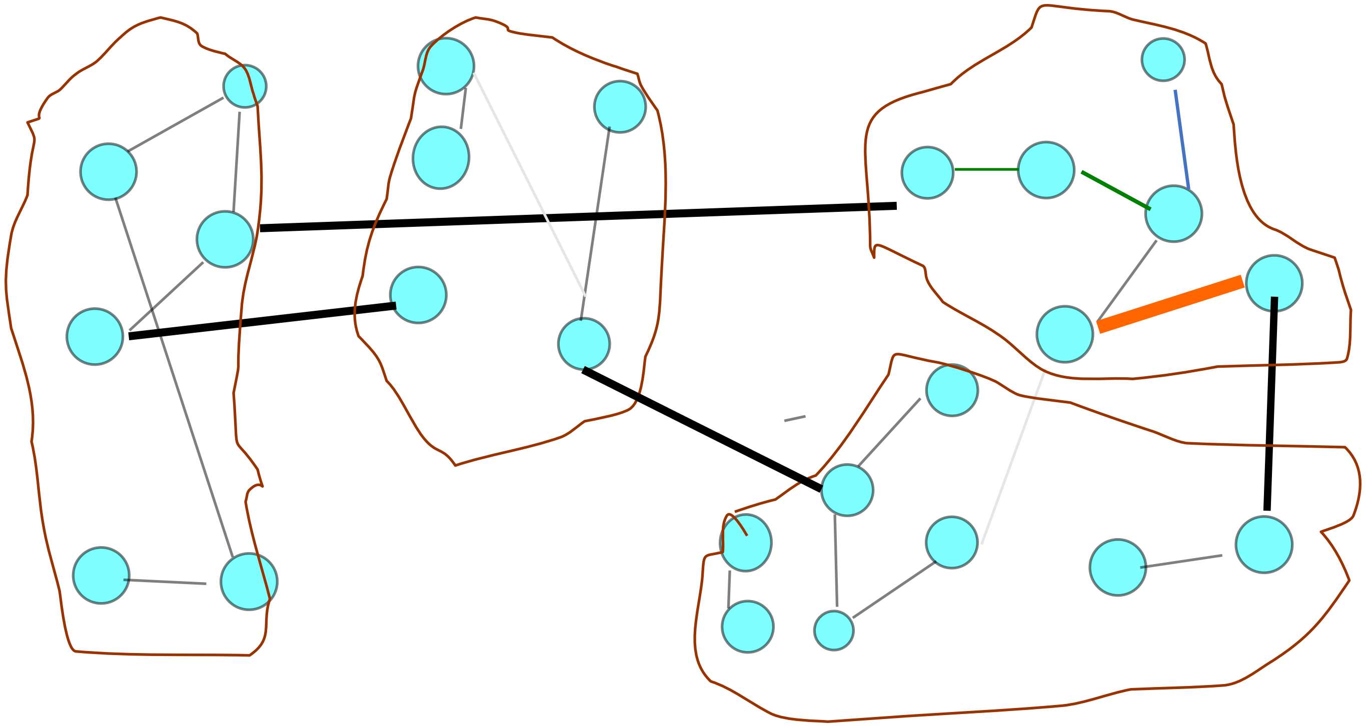
3. Hierarchical Clustering

Keep “external” edges betw each pair of connected clusters;
Spanning tree inside cluster; spanning tree of clusters



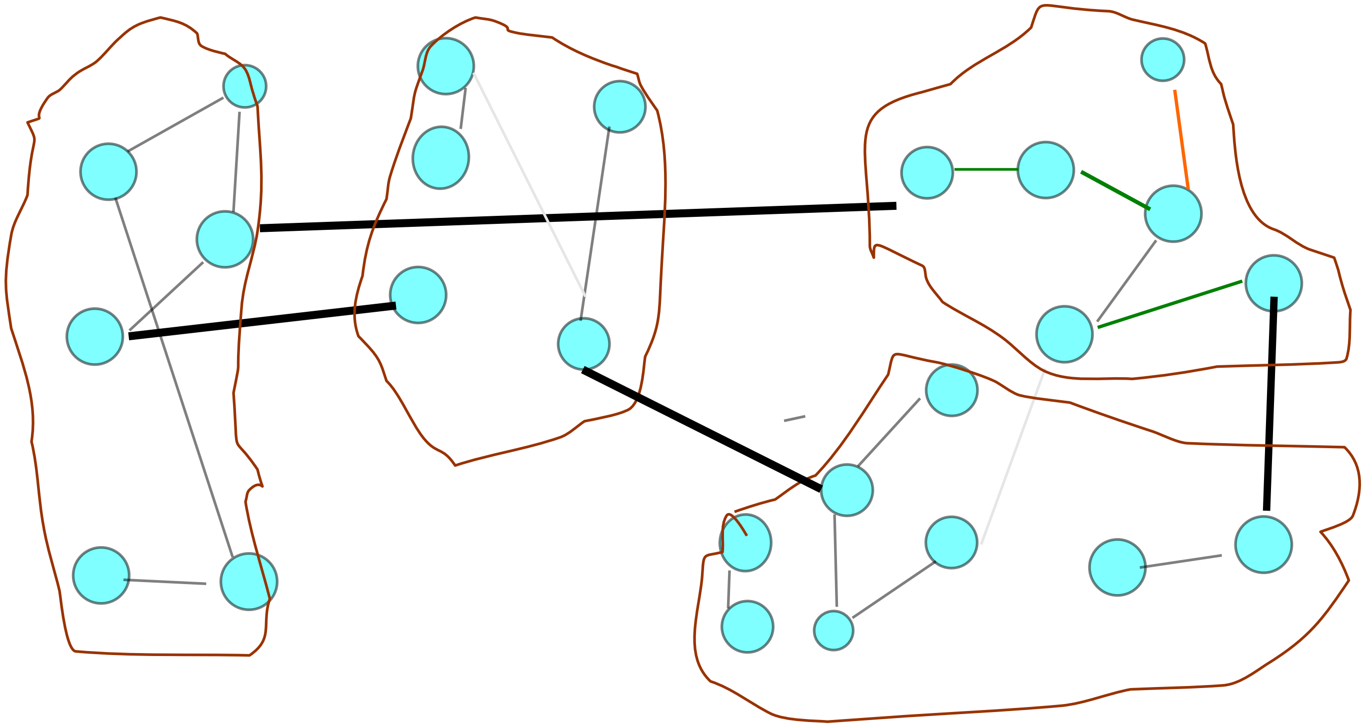
2. Hierarchical Clustering

If deleted external edge: find new external edge.
Find replacement external tree edge if needed



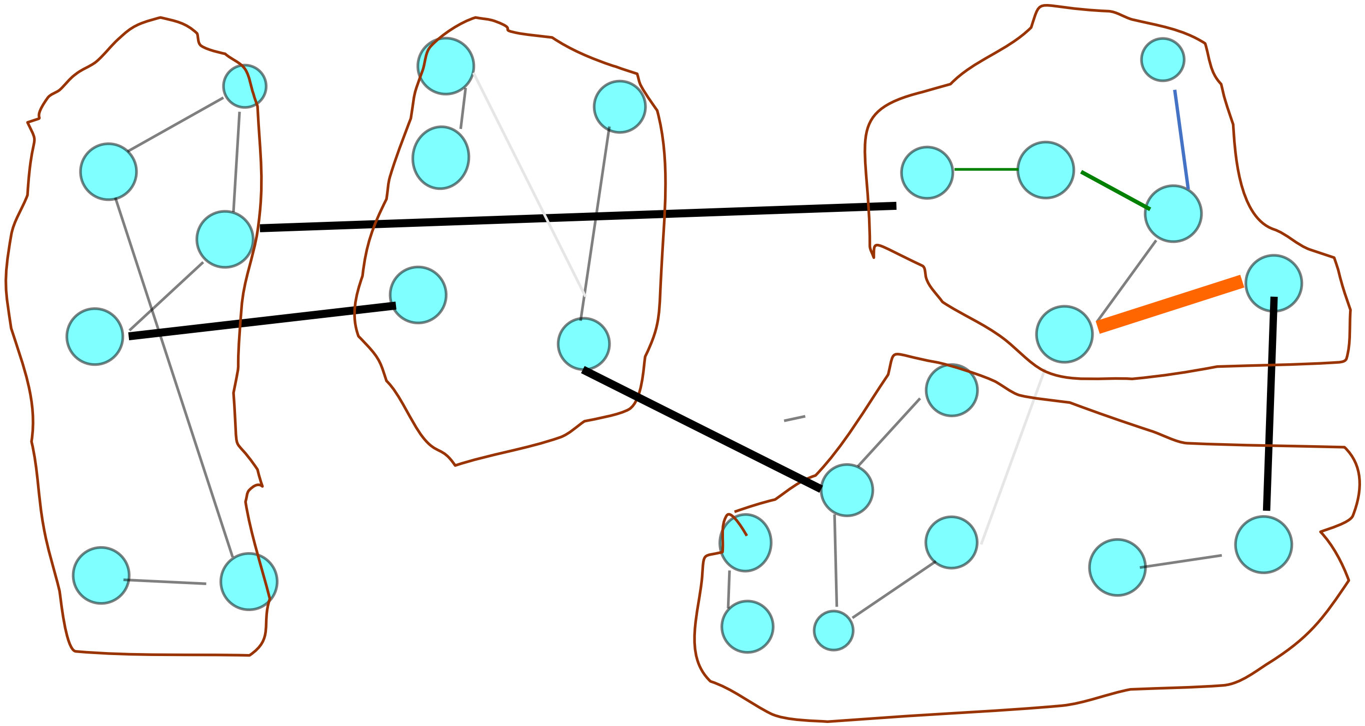
2. Hierarchical Clustering

- $m^{2/3}$ total edges inside each cluster
- $m^{2/3}$ selected external edges



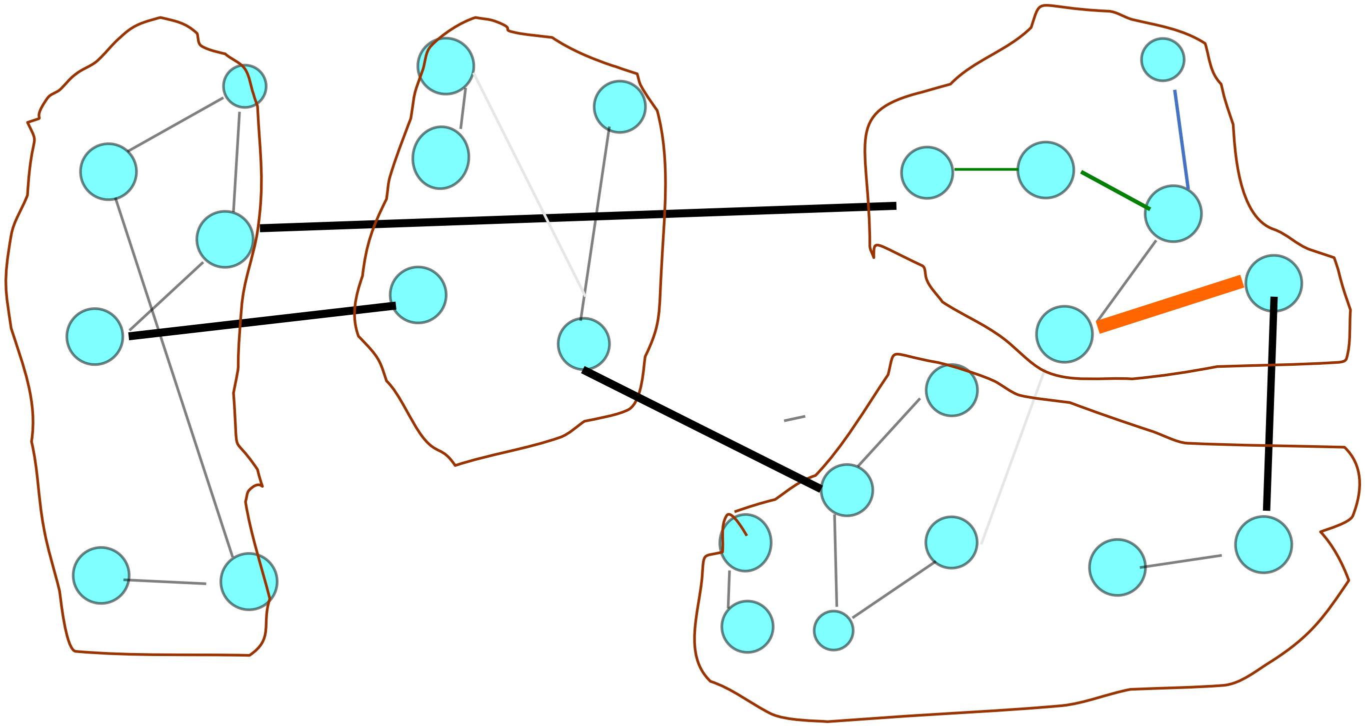
3. Hierarchical Clustering

Delete internal edge→: check inside edges first or split cluster.
Merge with neighbor cluster if cluster is too small.



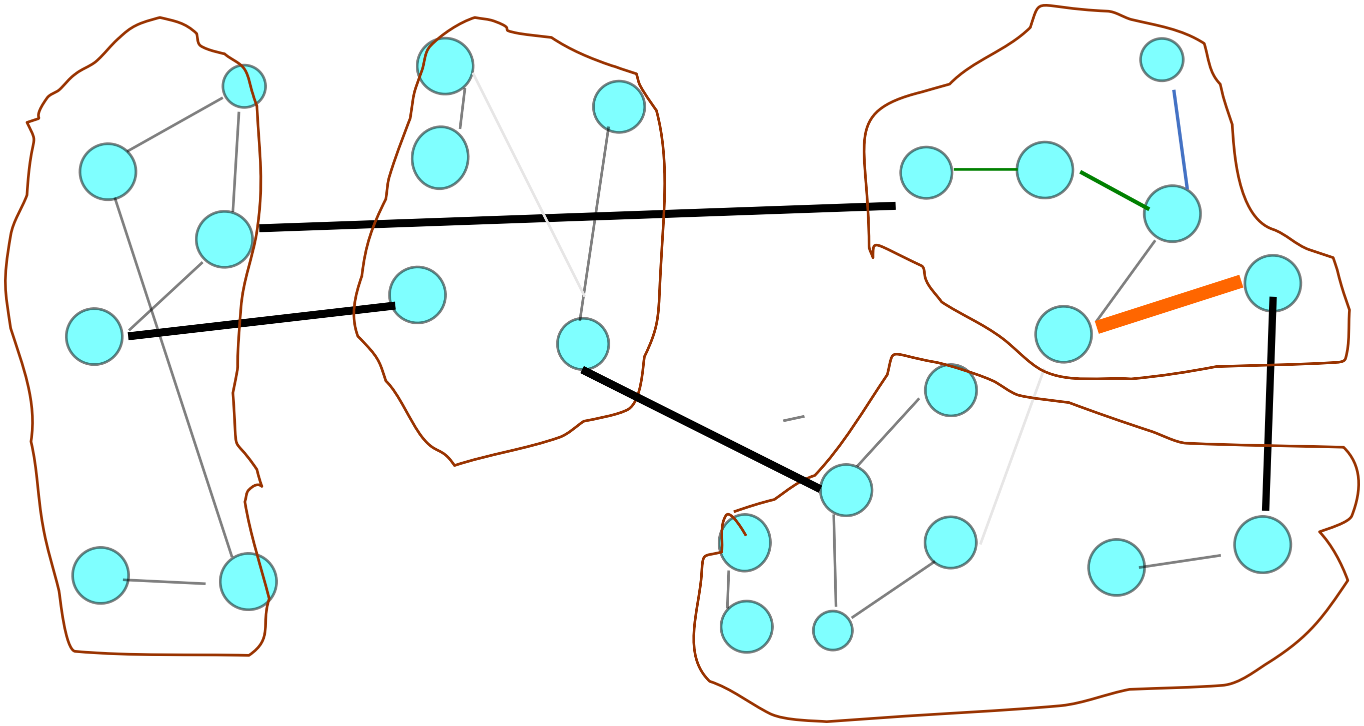
3. Hierarchical Clustering

Hierarchical decomposition, 30 pages later→√m



3. Hierarchical Clustering

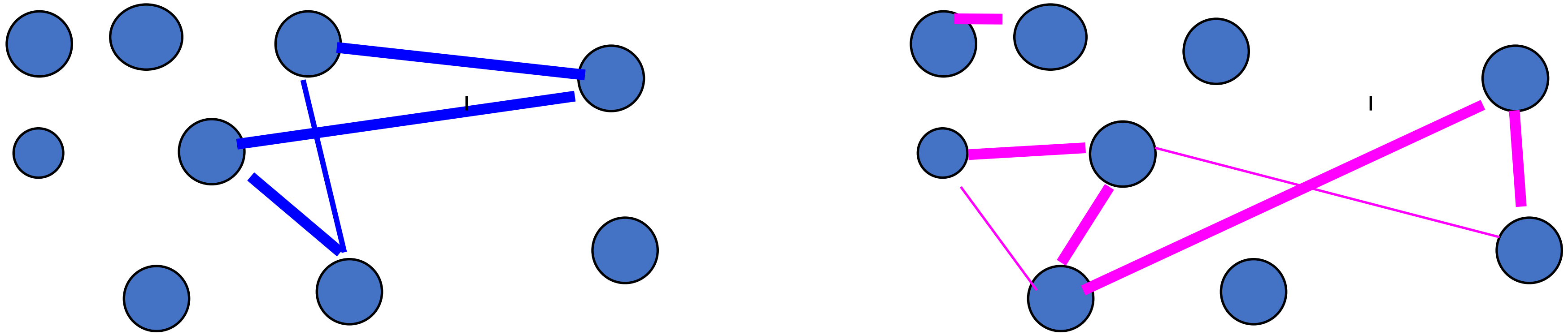
Gave rise to simpler hierarchical trees:
TOP trees, RC trees, parallel RC trees.



Idea 4 : Sparsification

Partition the edges

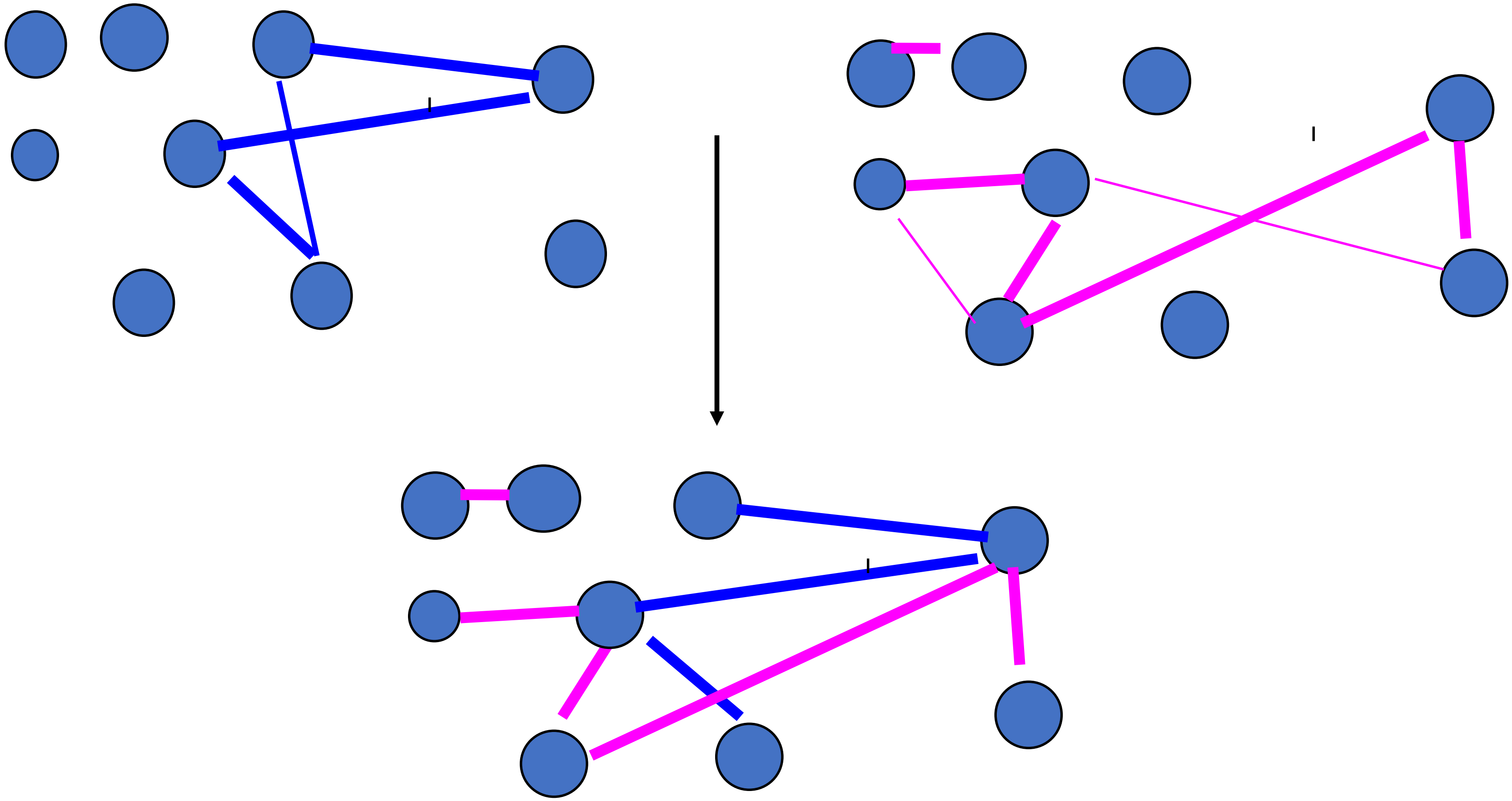
Determine spanning forest of each= sparse certificate



Idea 4: Sparsification

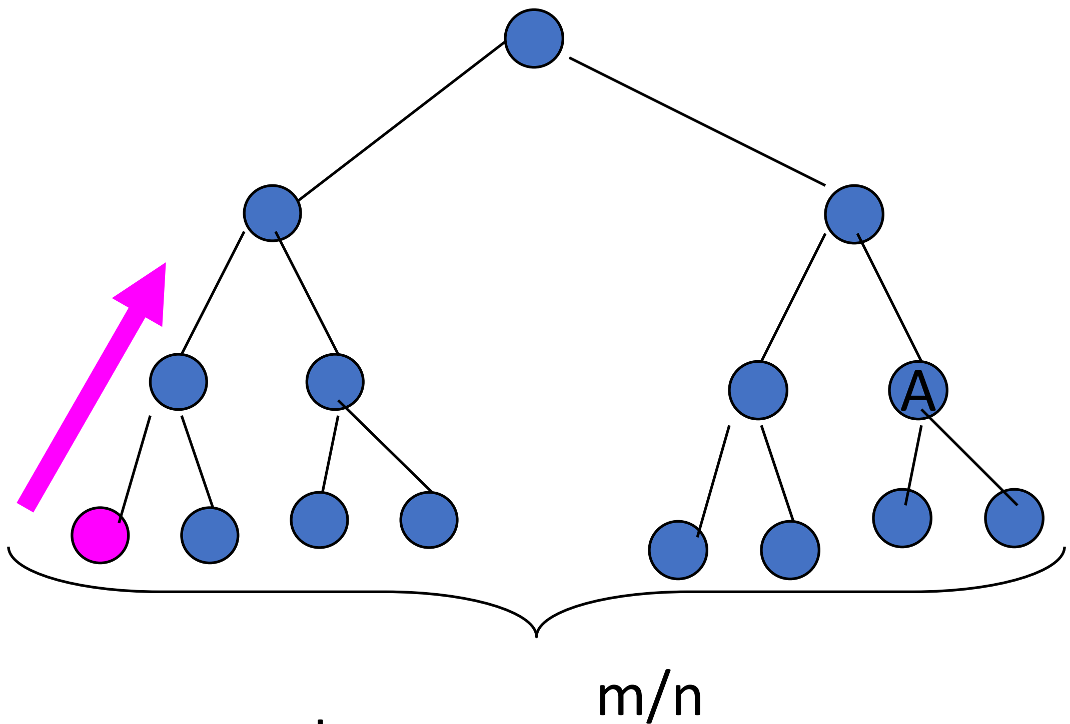
Partition edges

Union of spanning forests contains spanning forest of union



Idea 4. Sparsification

Parent graph = union of spanning forests of children

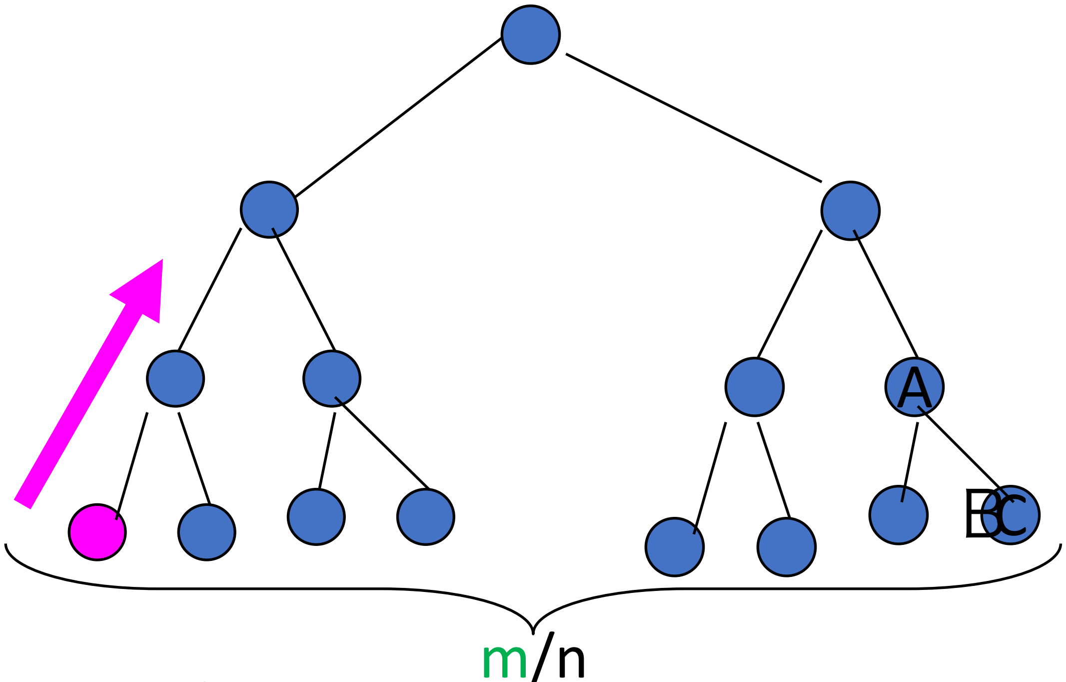


Propagate up change

Update time= (log m/n)*(cost for m<2n)

Idea.4: Sparsification

Parent = union of sparse certificates of children



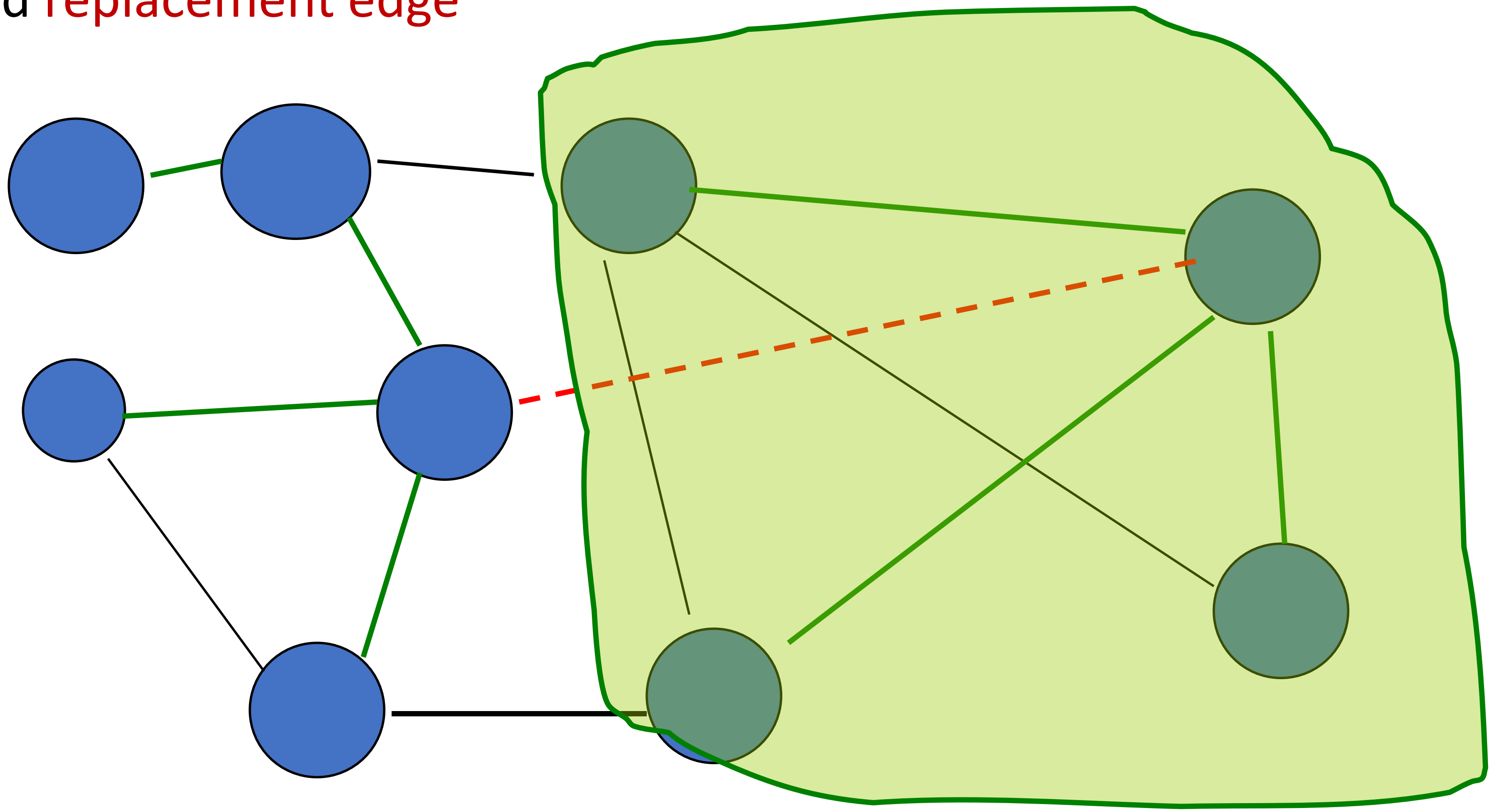
Propagate up change

Update time ~cost for graph of m<2n

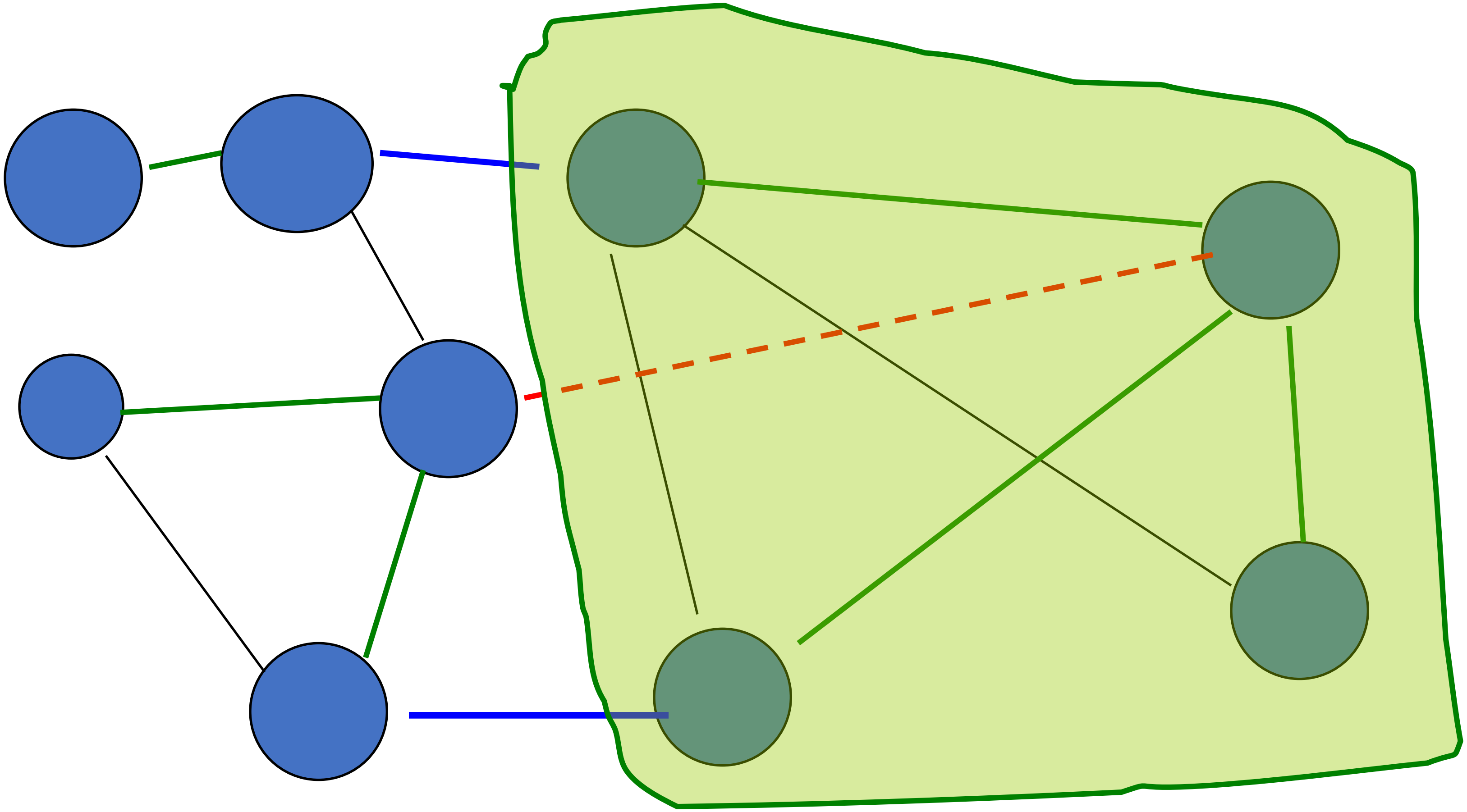
-->Only graphs of size 2n-2 ever need to be considered!

Idea 5: Sampling for dynamic decomposition

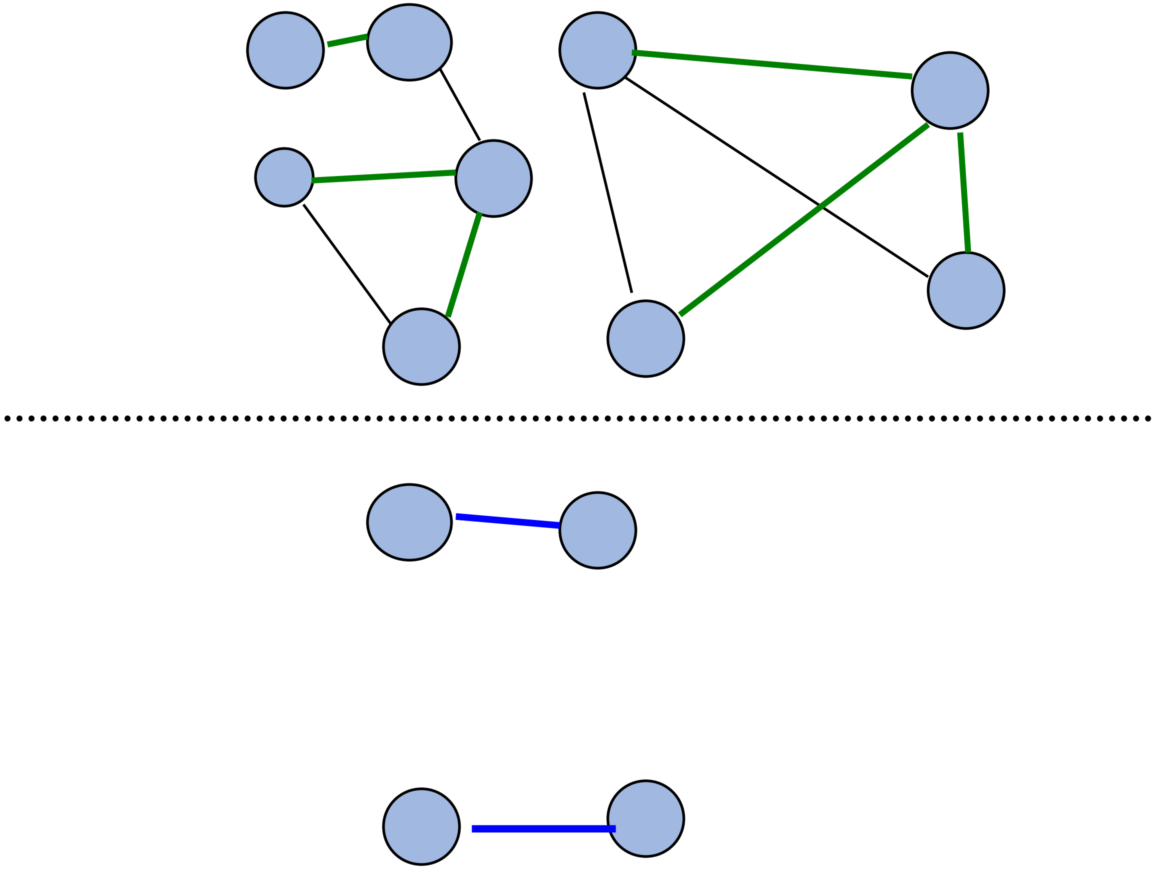
When a tree edge is deleted, randomly sample nontree edges incident to the smaller component to find replacement edge



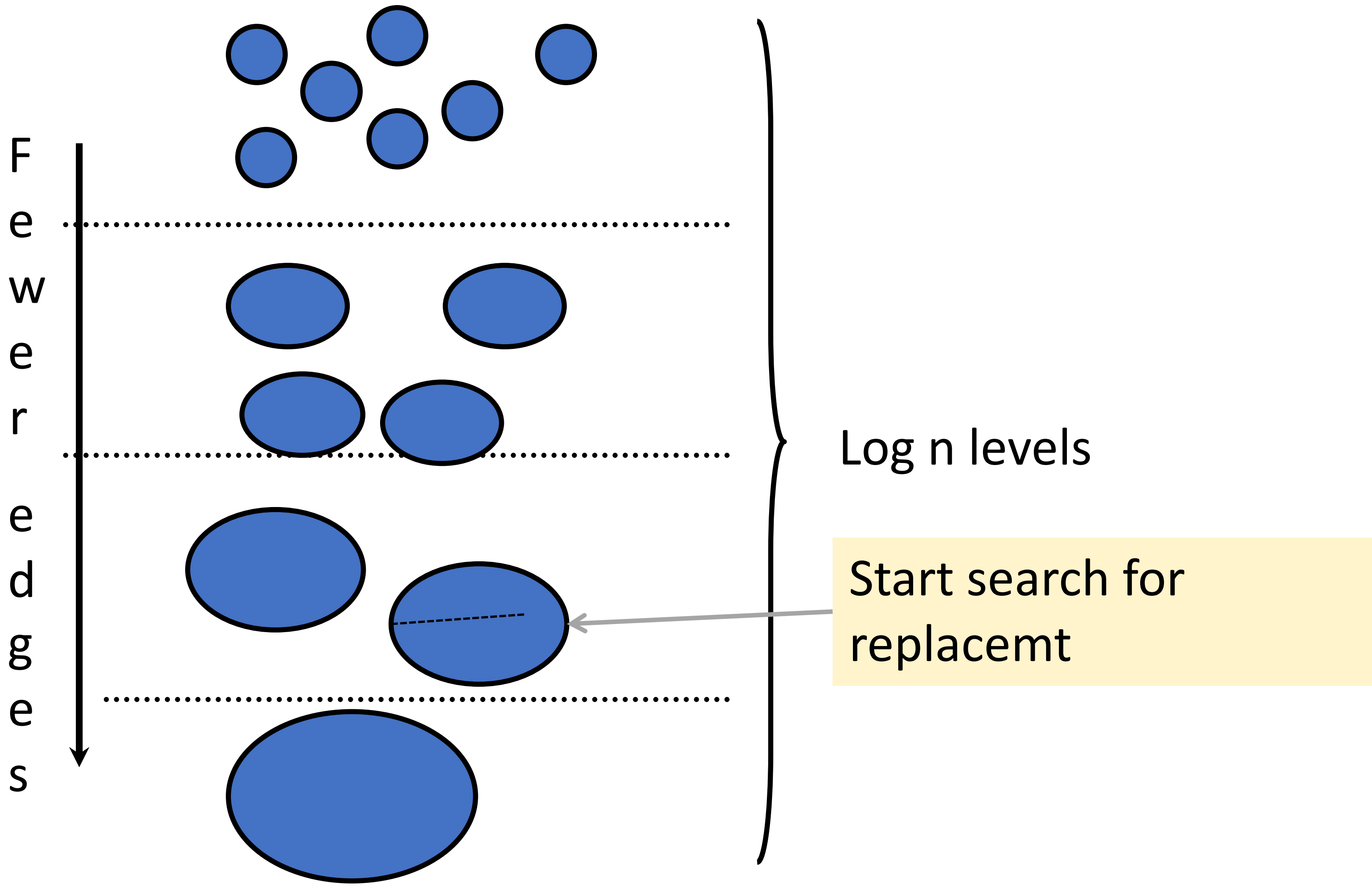
Else the cut is SPARSE
Check ALL the edges incident to the smallest component;
Move edges in cutset down to a “lower level”.



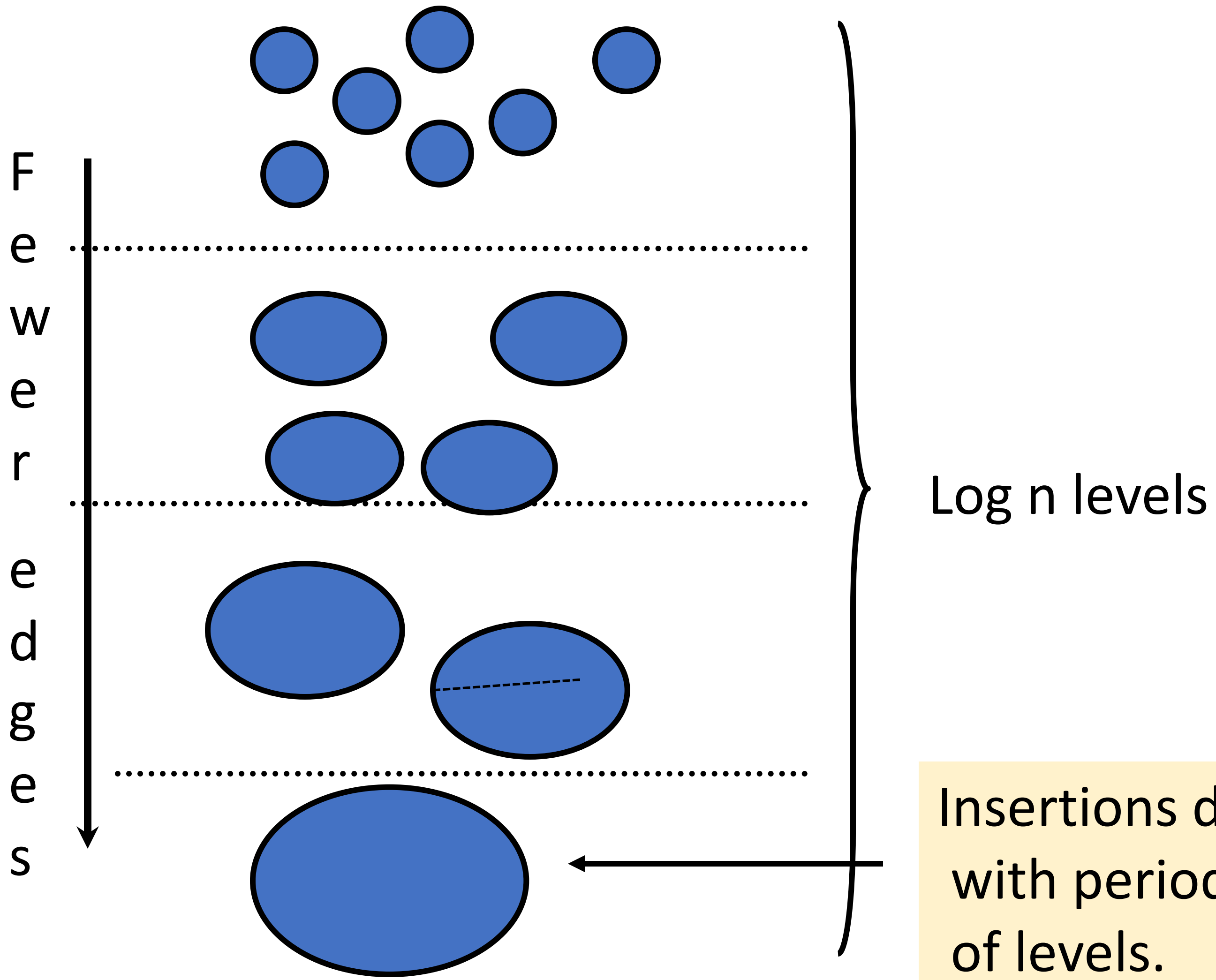
Cost per level =cost of searching smaller components + sampling cost



Each edge looked at log n times per level (from idea (2))
+ sampling cost.



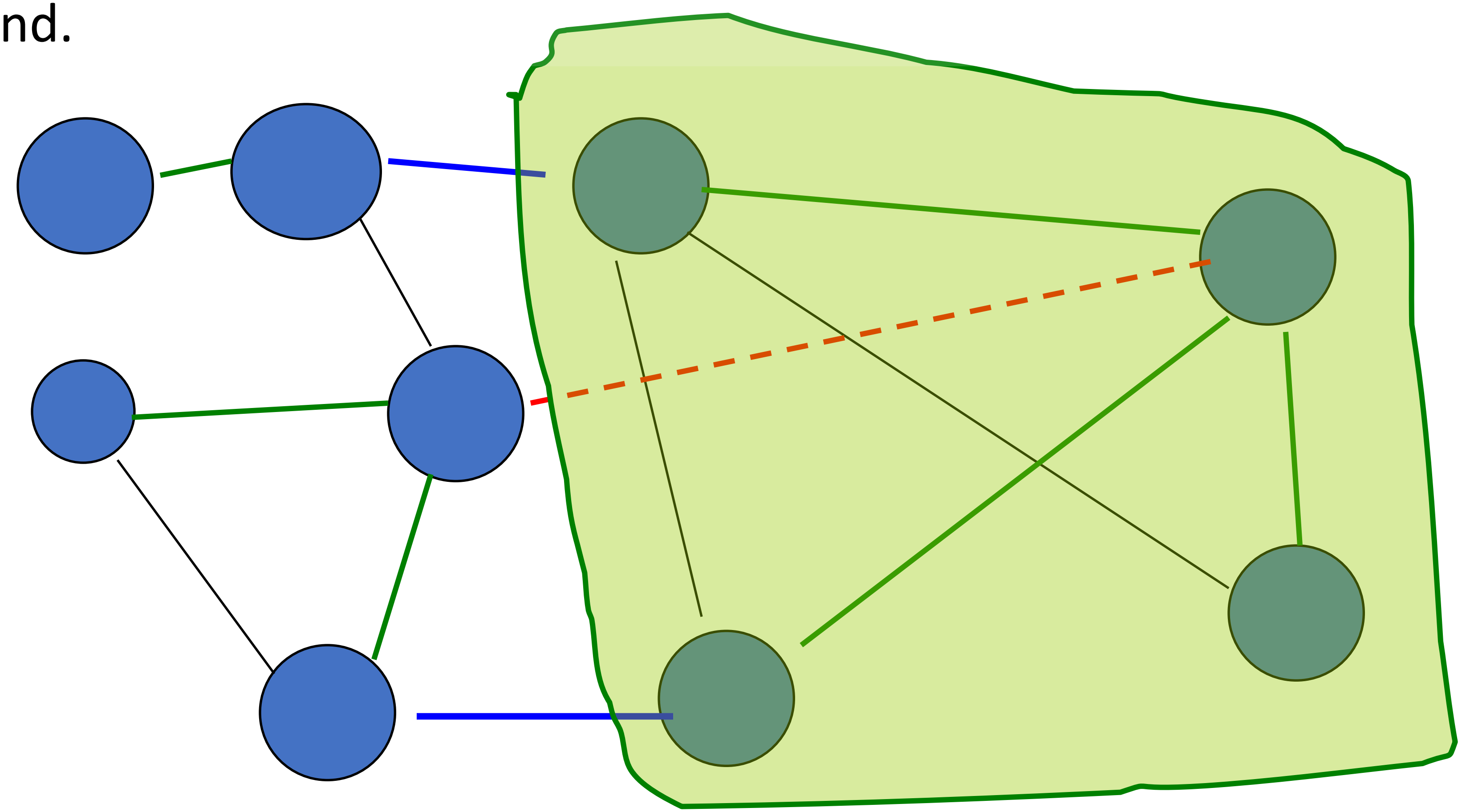
Each edge looked at log n times per level (from idea (2))
+ sampling cost.



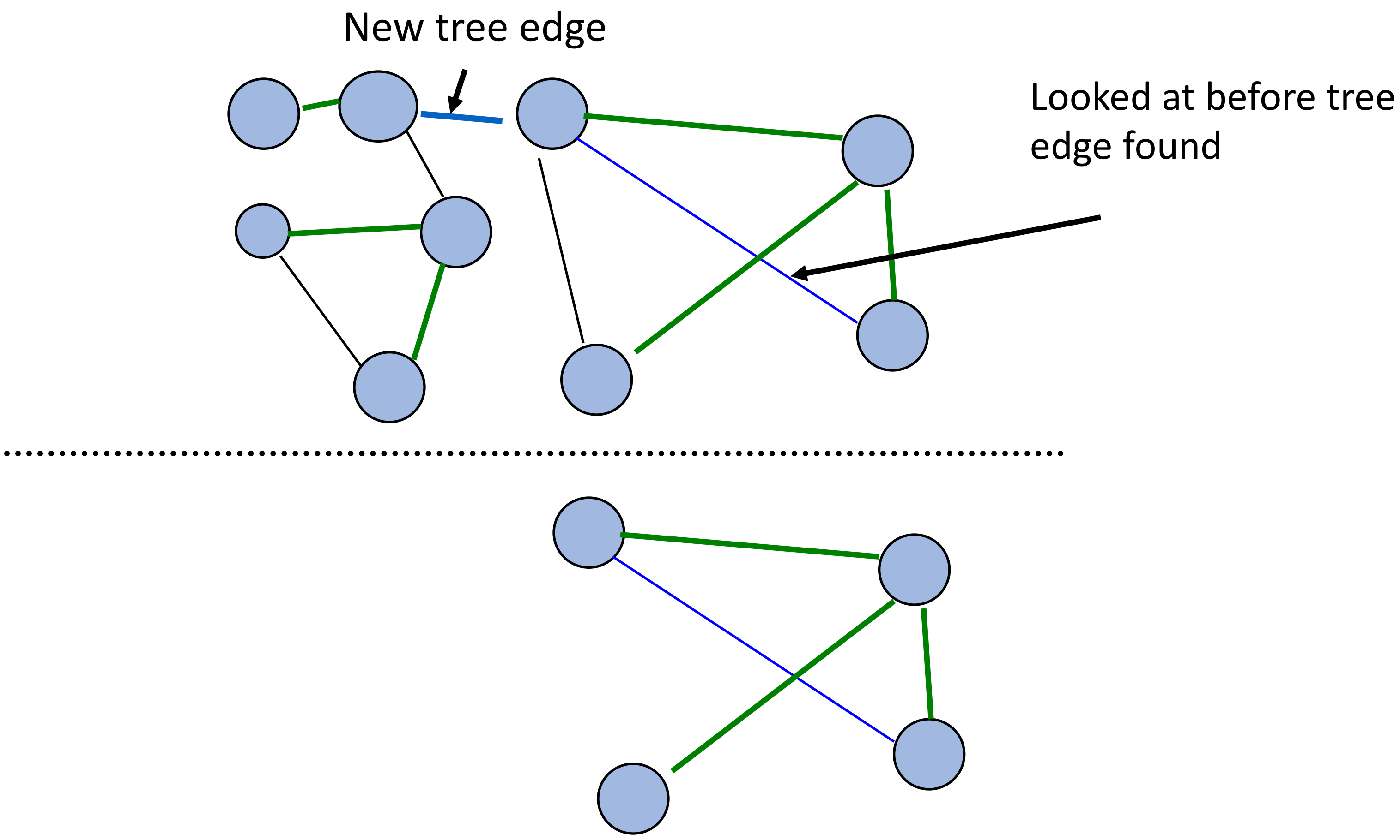
Insertions done here,
with periodic rebuilds
of levels.

IDEA 6: Deterministic dynamic decomposition (HDT)

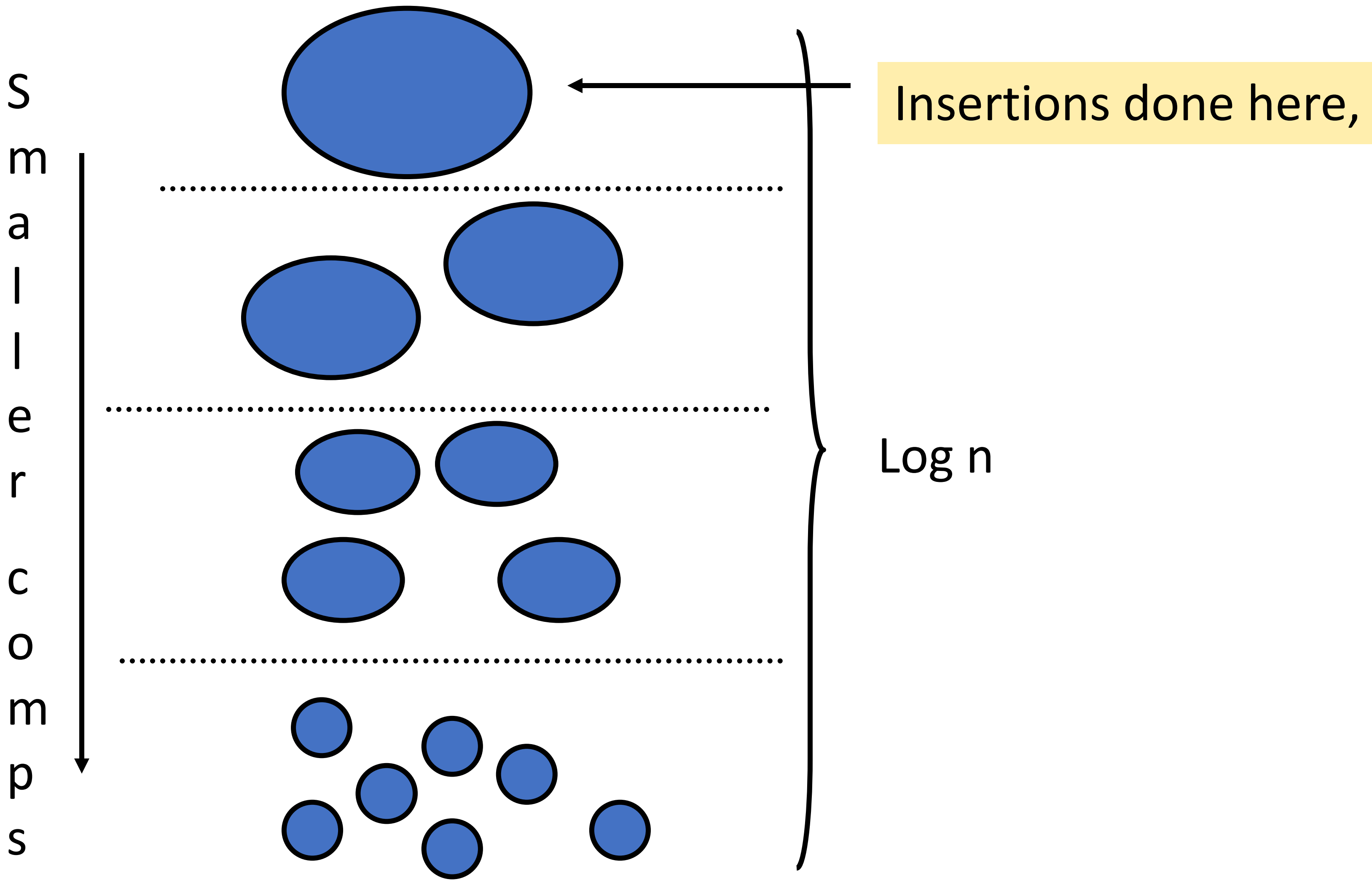
Look at each edge in smaller component until replacemt edge is found.



Move edges which were looked at which are NOT in the cutset to a lower level



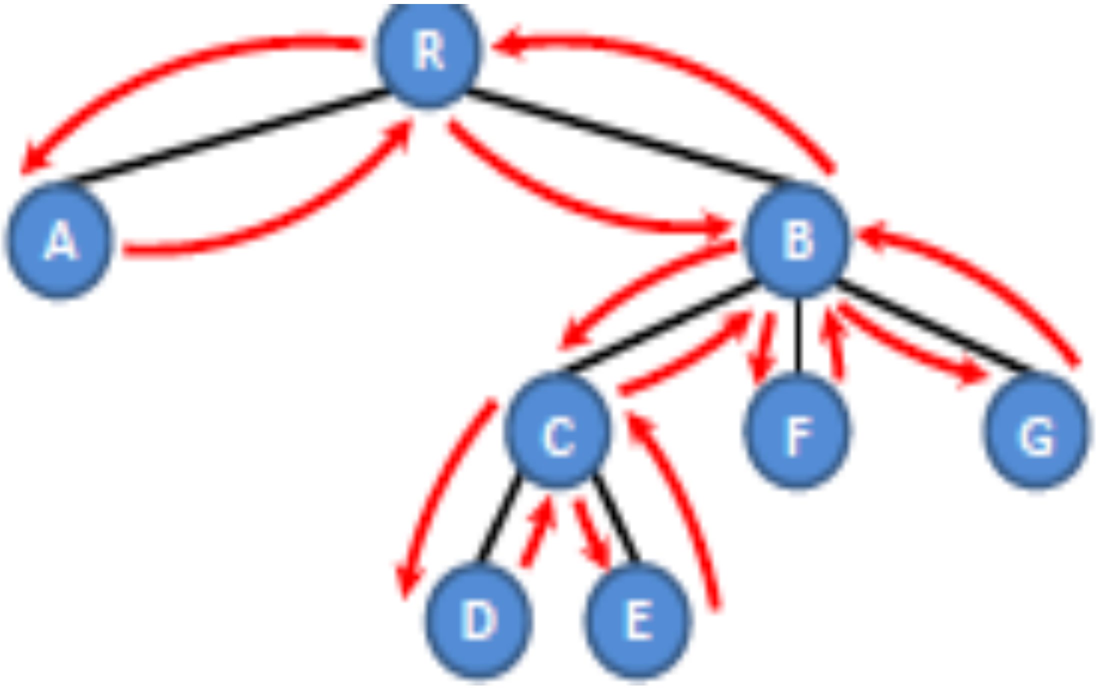
Each edge looked at no more than log n times.



IDEA 7: "ET-TREES"

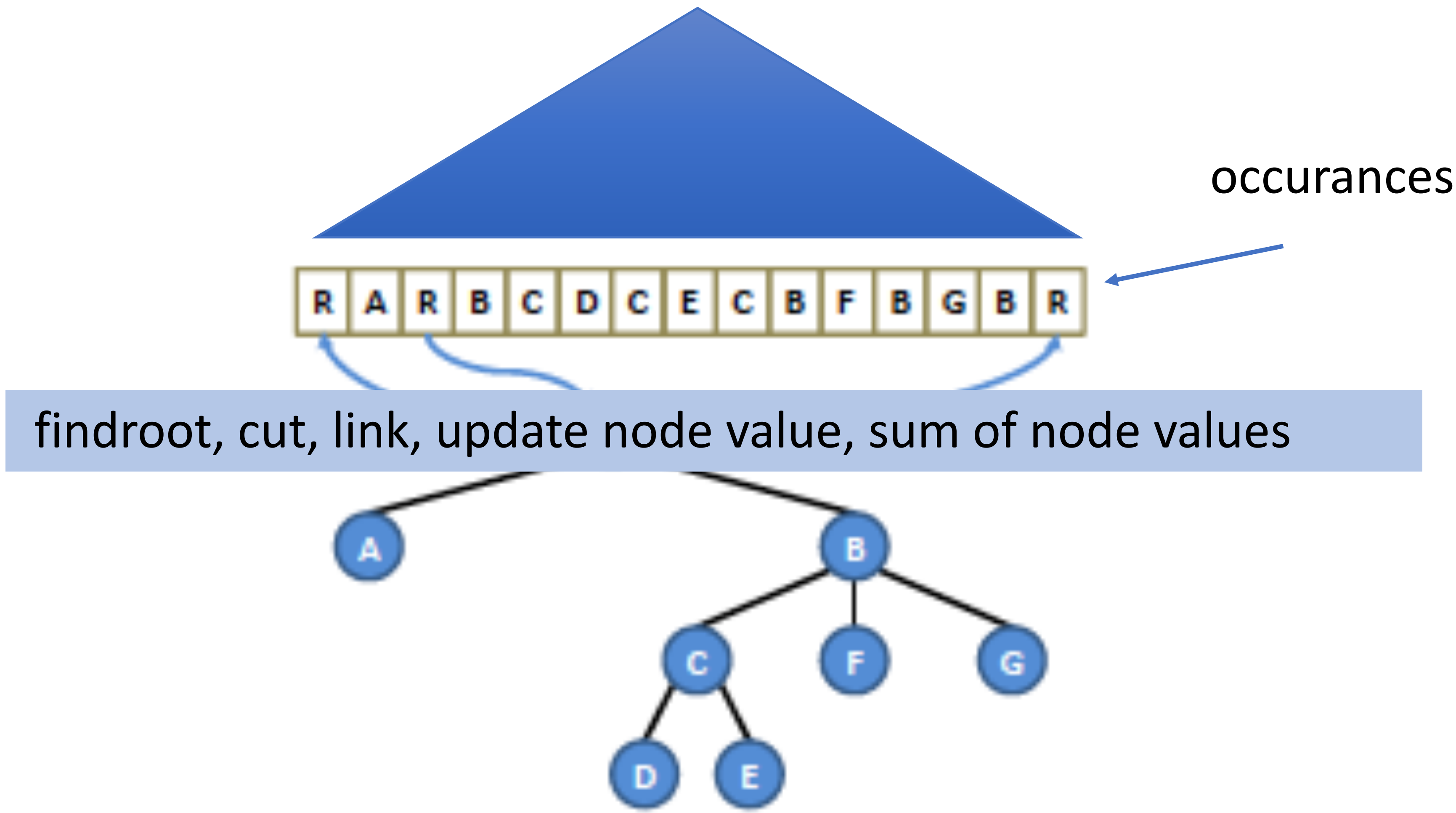
Dynamic decomposition introduces new, simple data structure to support subtree queries, random sampling

IDEA 7: "ET-TREES"



IDEA 7: “ET-TREES”

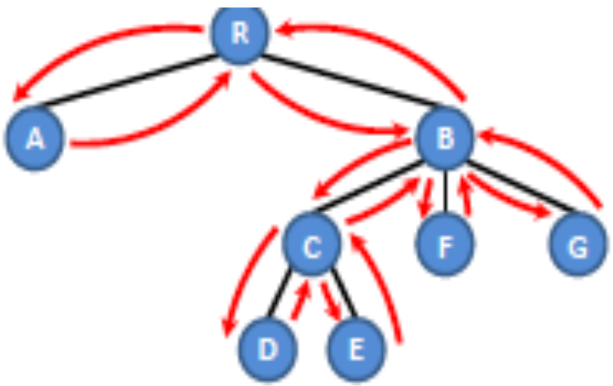
stored in augmented balanced search tree



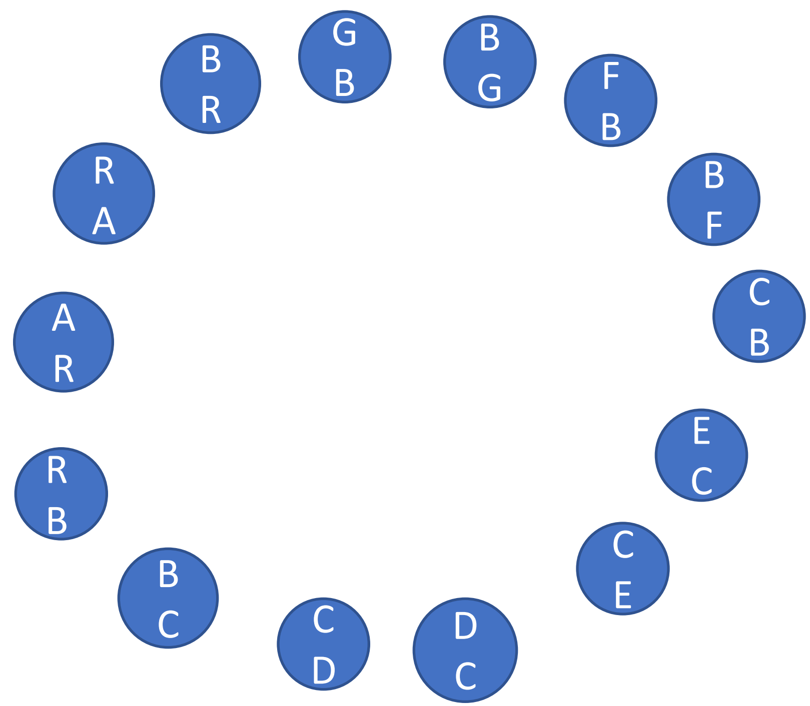
IDEA 7: "ET-TREES"

Batch parallel implementation of ET trees

Batch parallel insert, delete, query (2019-Tseng, Dhulipala, Blelloch)



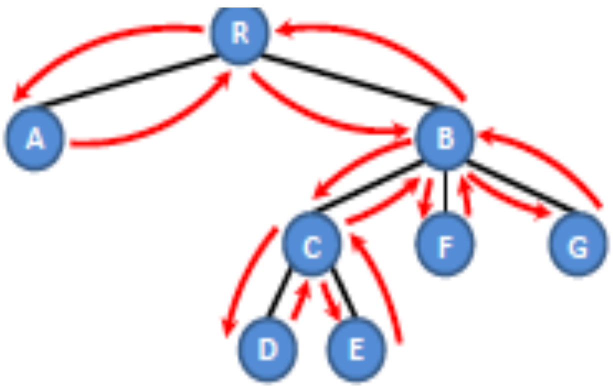
Implemented as unrooted circular skip list of directed edges



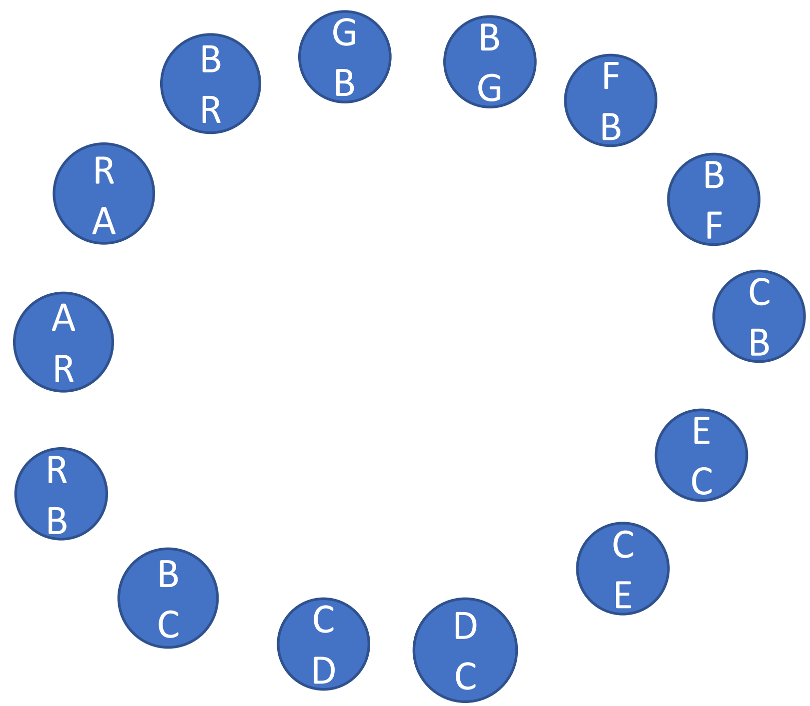
O(k log (1+n/k)) exp work,
O(log n) depth w.h.p.

Batch parallel implementation of HDT

Batch parallel insert, delete, query (2019- Acar, Anderson, et al.)



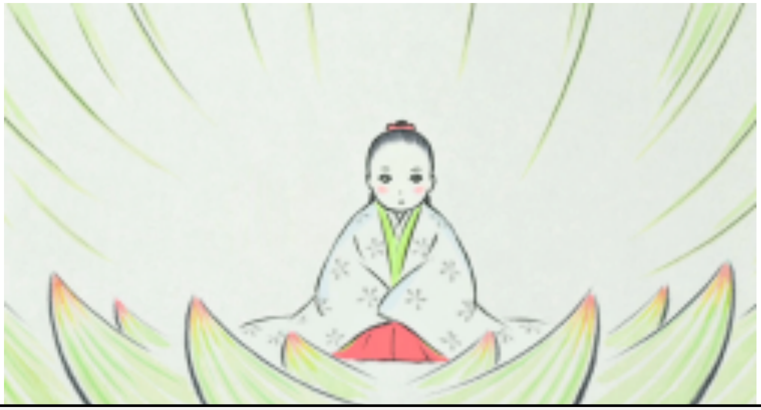
Implemented as unrooted circular skip list of directed edges



O(k log n log (1+n/k)) exp amortized work

O($\log^3$ n) update depth w.h.p.

Idea 8: XOR method



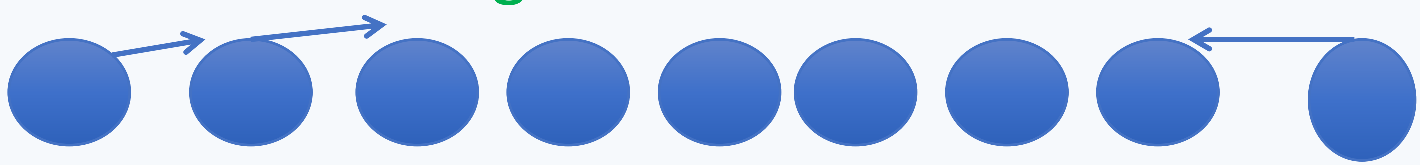
Observe: Let T any subset of V

If |cutset (V\T, T)|=1 then
Parity(Sum of degrees of nodes in T) =1

Solves "CUTSET" PROBLEM

Given a dynamic forest of disjoint trees T in G w/constant prob., find an edge in the cutset (T,V\T) for each T

Updates: insert edge, delete edge, make tree edge, make non-tree edge.



Find_edge(T) returns an edge in E in the cutset of (T, V\T)

Idea 8: XOR method

To solve cutset problem:

- Each edge (a,b) has unique binary name <ab>
- For each node **a**, let
- **v(a)**=bitwise XOR (names of edges incident to node **a**)

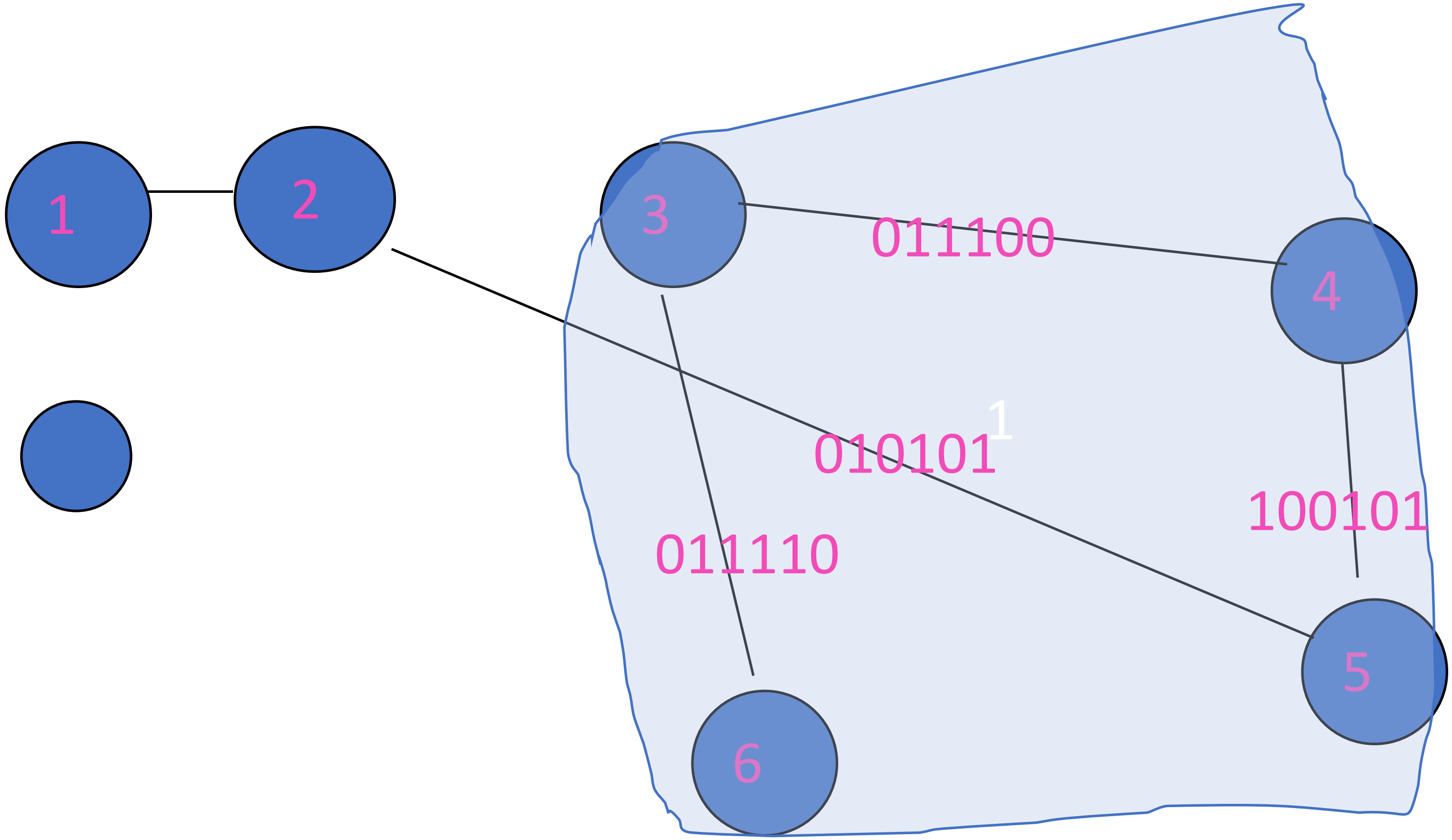
IF there is exactly one edge {a,b} in cutset of (T,V\T) then

$XOR_{a \text{ in } T}\ v(a) = \langle a,b \rangle$

(all other names cancel out)

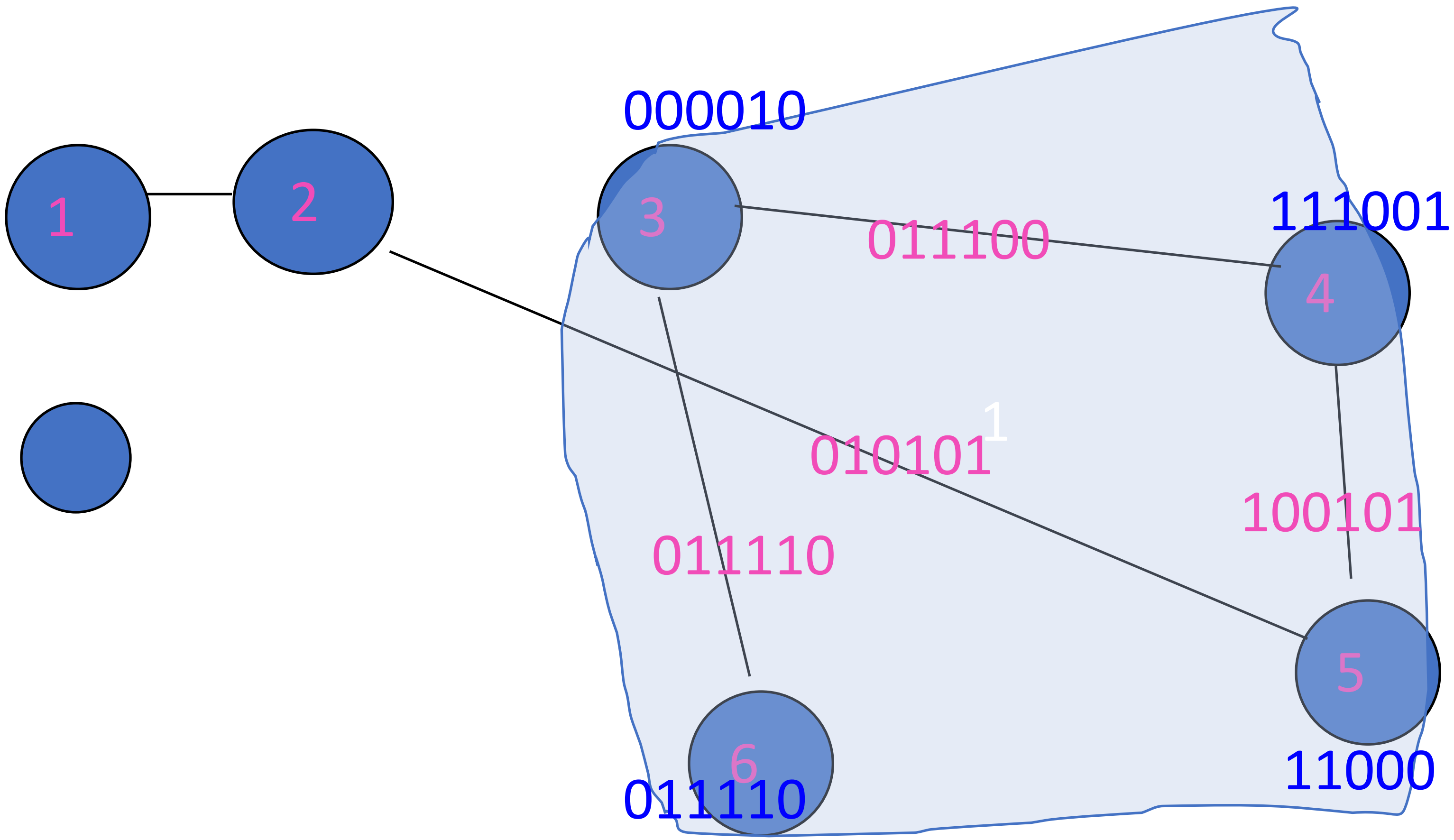
Idea 8: XOR method

Example:



Idea 8: XOR method

Sums at nodes: 010101=(2,5)



What if $|\text{cutset}(T, V \setminus T)| > 1$?

Maintain sample sets of edges in G:

- For i=0 to $\log\binom{n}{2}$

  Pr(edge in Sample i)= $\frac{1}{2}^{i}$
- For each node:
  - $\mathbf{v_i(a)}$= XOR(<a,b> in Sample i)

Idea 8: XOR method

What if |cutset (T,V\T)|>1?

Maintain sample sets of edges in G:

- For i=0 to $\log\binom{n}{2}$

  Pr(edge in Sample i)= $½^i$
- For each node:
  - $\mathbf{v_i}(\mathbf{a})$= XOR(<a,b> in Sample i)

If T is not a spanning tree in G

- k <- max i s.t. $\text{XOR}_{a \text{ in } T}\, v_i\,(a) \neq$ <00..0>
- find_edge(T) returns $\text{XOR}_{a \text{ in } T}\, v_K\,(a)$

With constant prob,

- find_edge(T) is an edge in the cutset and
- k= $\lceil log|cutset(T, V\backslash \text{T})|\rceil$

Idea 8: XOR method

What if |cutset (T,V\T)|>1?

Maintain sample sets of edges in G:

- For i=0 to $\log\binom{n}{2}$

  Pr(edge in Sample i)= $\frac{1}{2}^{i}$

- For each node:
  - $\mathbf{v_i}$(**a**)= XOR(<a,b> in Sample i)

If T is not a spanning tree in G

- k <- max i s.t. $XOR_{a \text{ in } T}\, v_i(a) \neq$ <00..0>
- find_edge(T) returns $XOR_{a \text{ in } T}\, v_K(a)$

With constant prob,

- find_edge(T) is an edge in the cutset and
- k= $\lceil log|cutset(T,V\backslash T)|\rceil$

Used to approximate cutset size

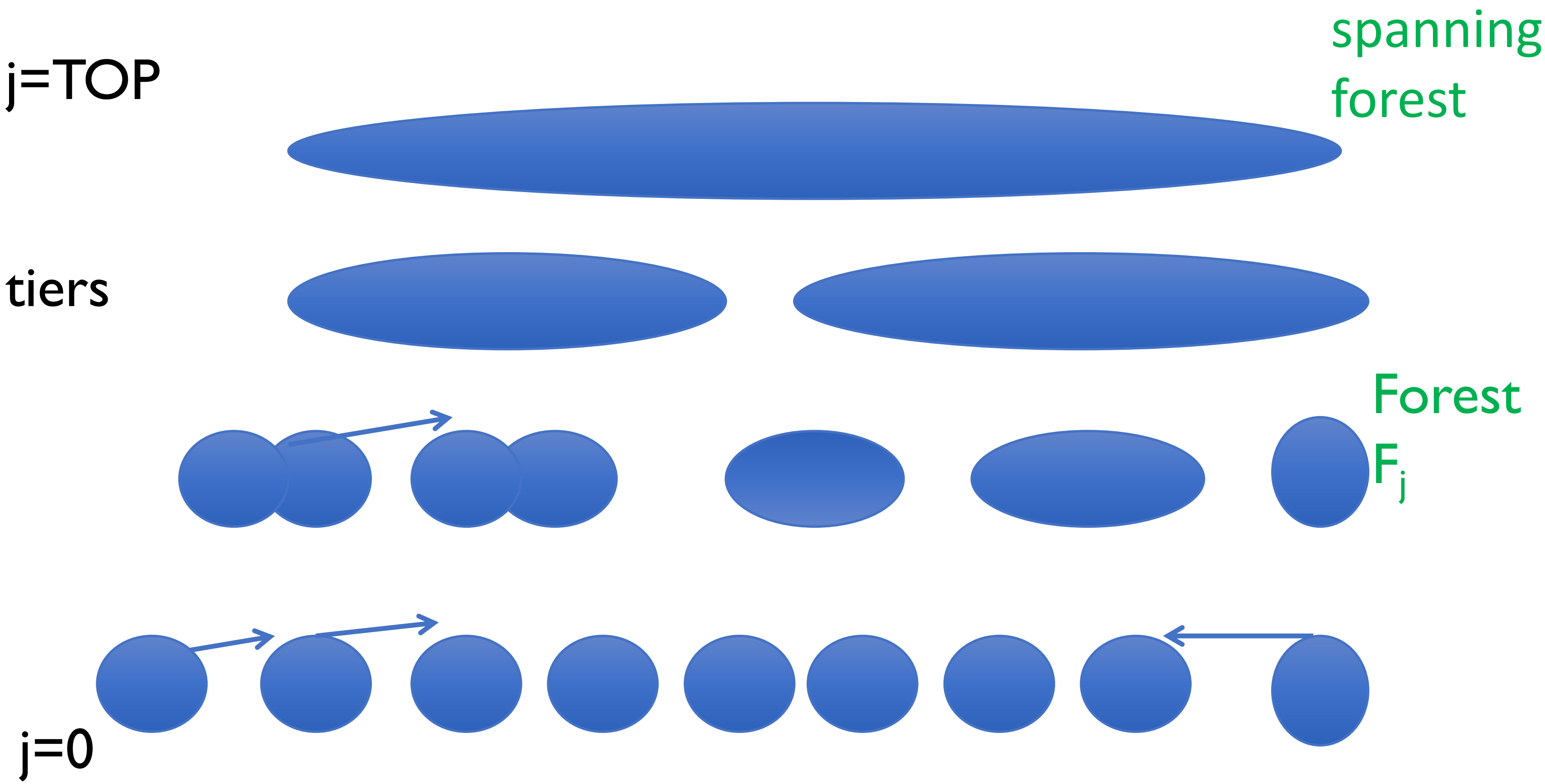
Solution to dynamic connectivity?? (not quite)

- Updates must be oblivious to random bits
- Analysis assumes no dependence between tree structure and random bits needed to find cutset edge

Idea 8: XOR method

- Boruvka type construction
  O(log n) Cutset data structures $c_0, c_1 \ldots c_{top-1}$ on G with forests $V = F_0 \subseteq F_1 \ldots \subseteq F_{top}$

with their own randomness



Idea 8: XOR method

query(a,b): Return True iff in F_{top} findroot(a) =findroot(b)

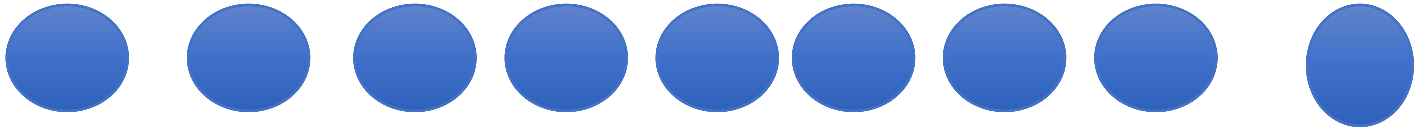
insert(a,b):

- insert edge {a,b} into cutset data structures $c_0, c_1 \ldots c_{top-1}$
- If query(a,b) =False, add {a,b} to $F_1 \ldots \subseteq F_{top}$ by
- calling c_i. make_tree_edge {a,b} for i=1 ... top

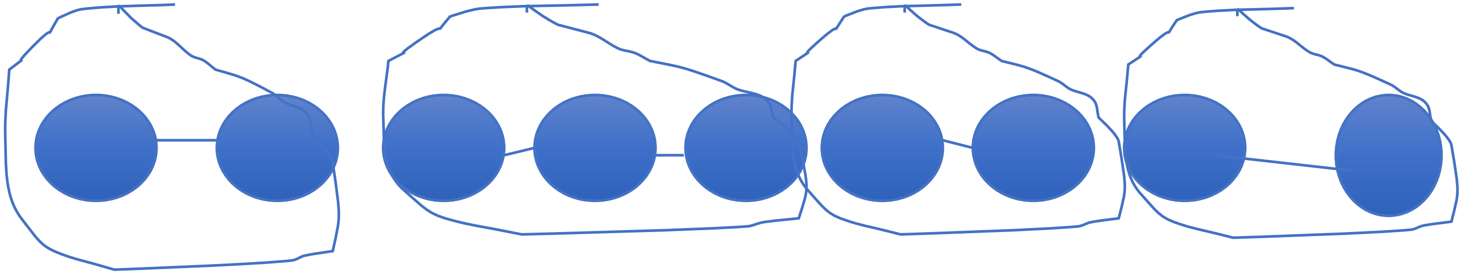
DEF: Let T_a denote tree in F_i containing node a

T_a is "unmatched" in F_i if T_a is no larger in F_{i+1}

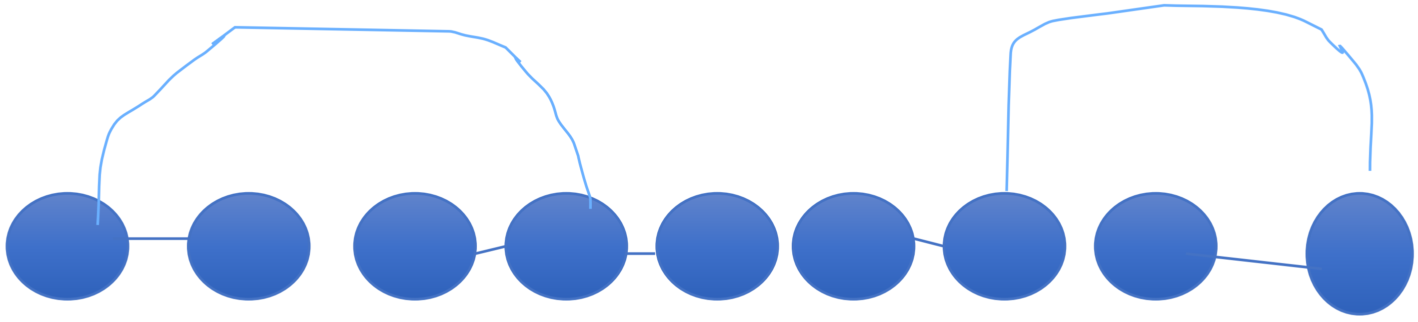
Build spanning forest in tiers



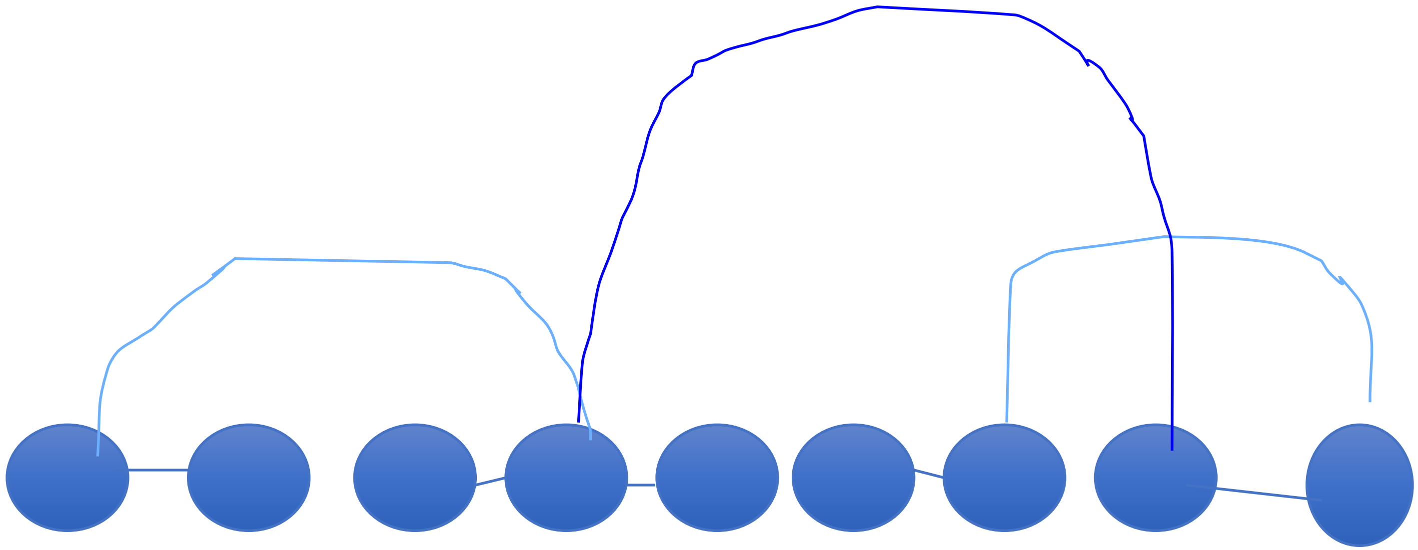
Build spanning forest in tiers



Build spanning forest in tiers

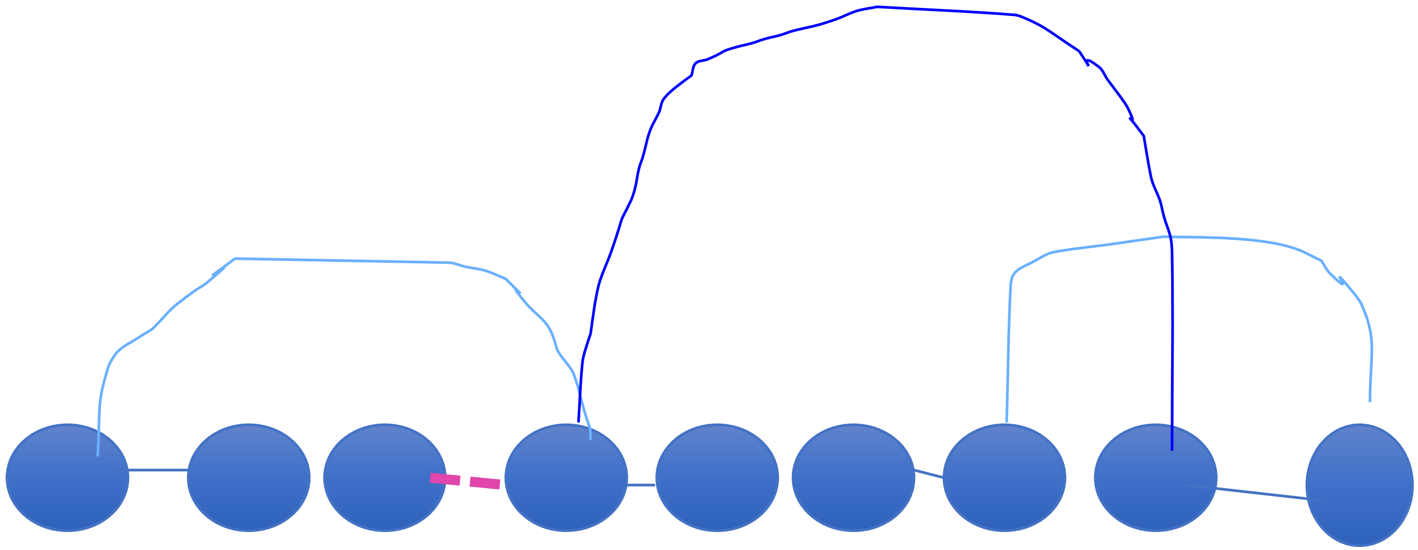


Build spanning forest in tiers

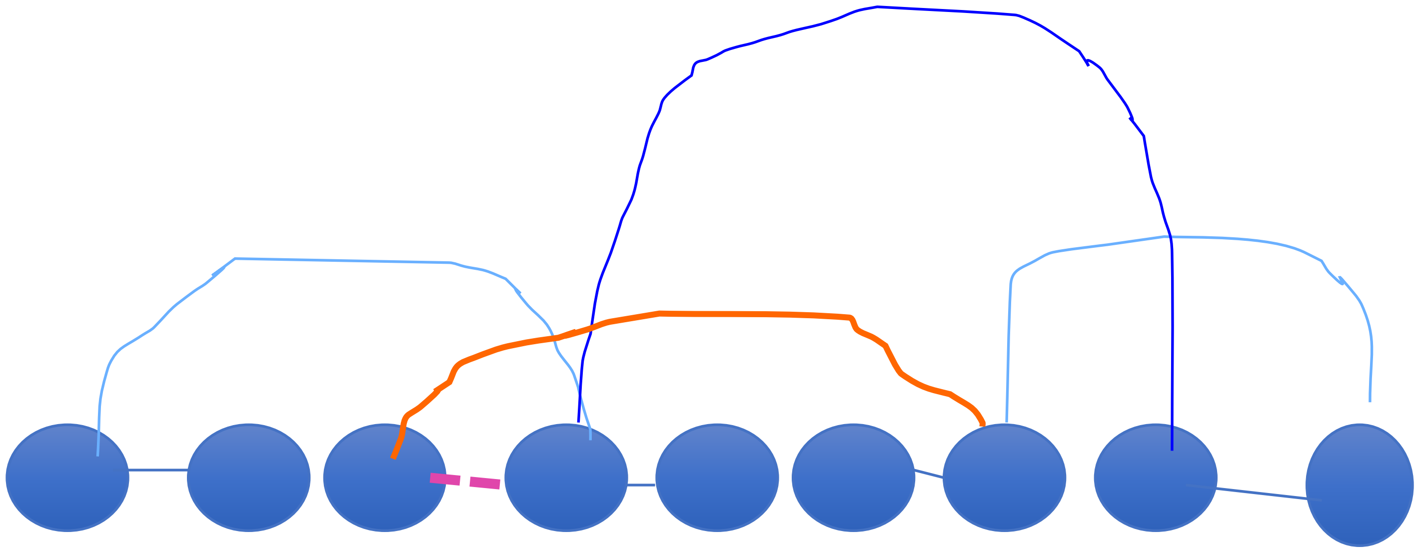


Idea 8: XOR method

Deletion:

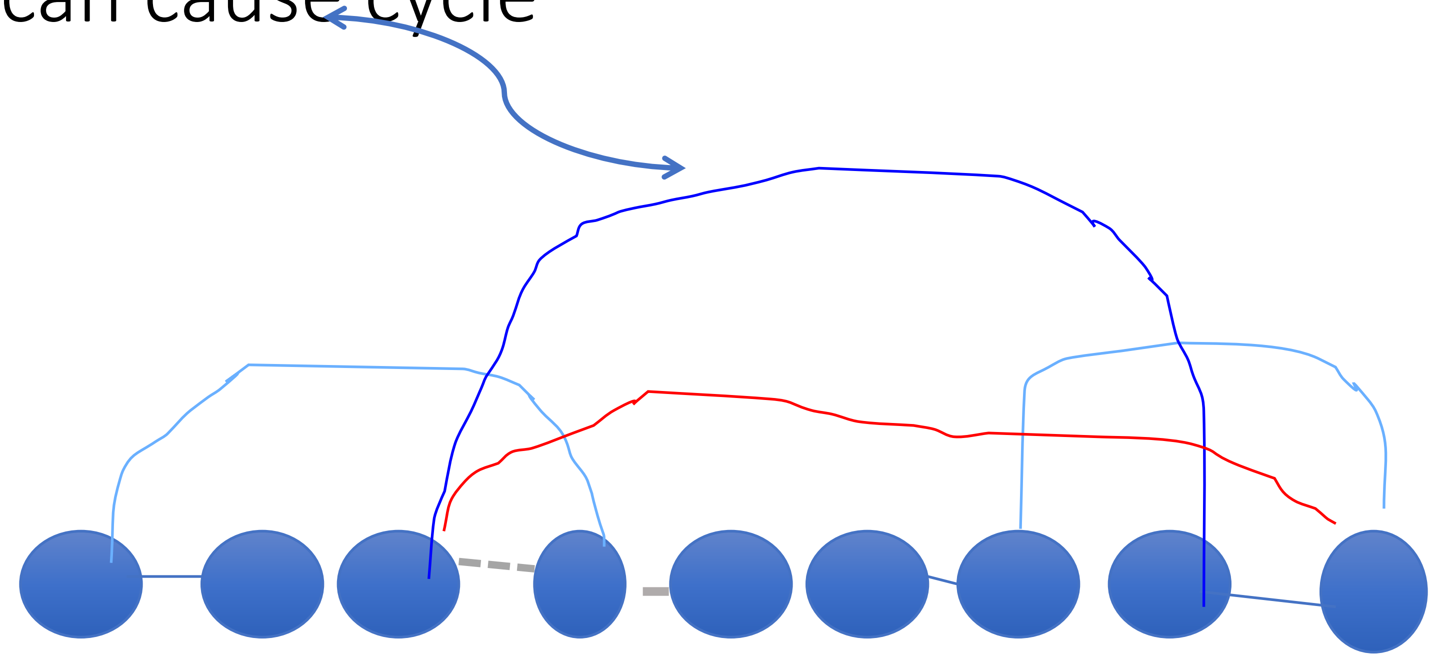


Deletion: If unmatched comp. find new edge

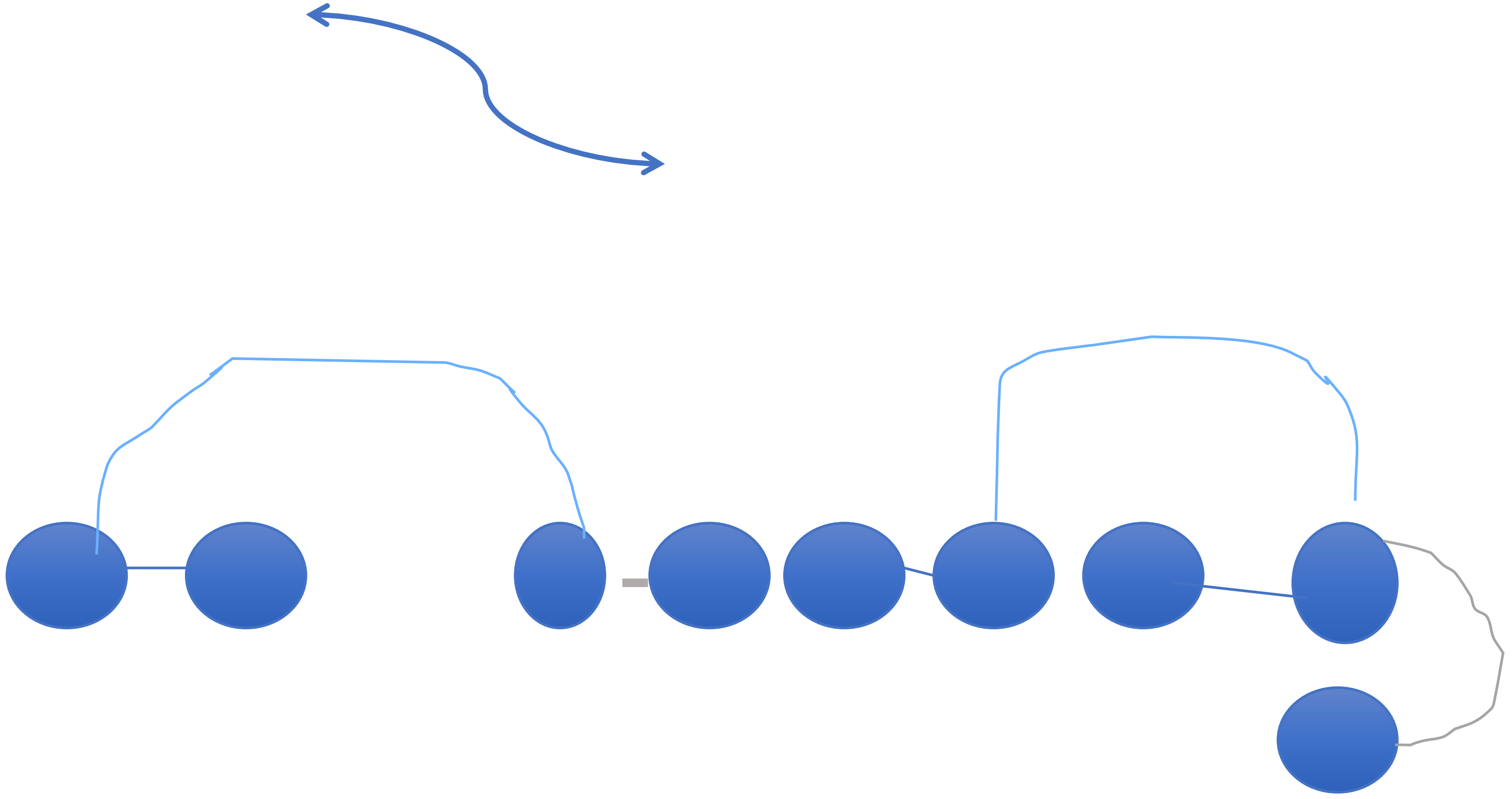


Idea 8: XOR method

Deletion: If unmatched comp. find new edge can cause cycle

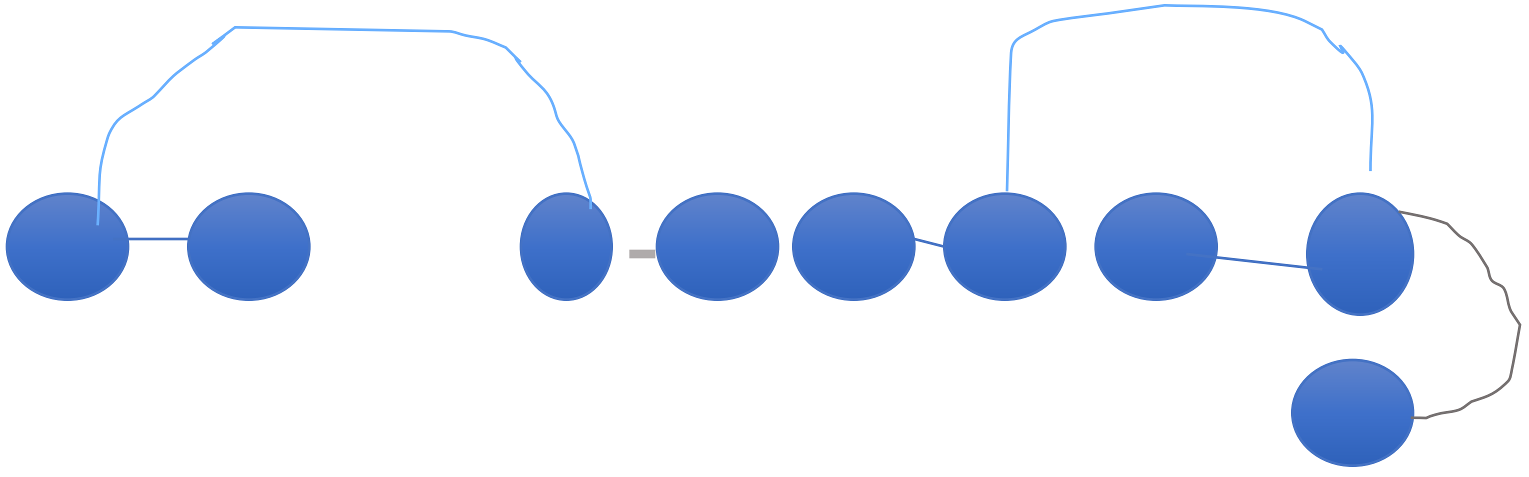


Deletion: cycle edge is not a tree edge



Idea 8: XOR method

Deletion:
cycle -> unmatched comp on higher level, etc.



Idea 8: XOR method

<u>delete(a,b)</u>

for i=1,..., top delete (a,b) from c_i

for a **then** b do

For i=1,..., top if (a,b) in F_i

If T_a in F_i is unmatched, (c,d) <-- c_i .find edge(T_a)

If (c,d) causes a cycle in some T in F_j

break cycle -- e' <--path_edge* (c,d) in $F_j \backslash F_{j-1}$

For k=j to top make_nontree edge(e')

For k=i to top make_tree edge(c,d).

*path_edge is not an ET-tree operation
Can afford $O(\log^2)$ time for its implementation w/no change to asymptotics

Can use link-cut trees, TOP trees, RC trees but these are harder to implement for arbitrary trees, especially in batch parallel model.

Part 2: Batch parallel KKM+, communication in distributed computing

Batch parallel implementation of KKM (Cann, K 2023)

TOOL BOX

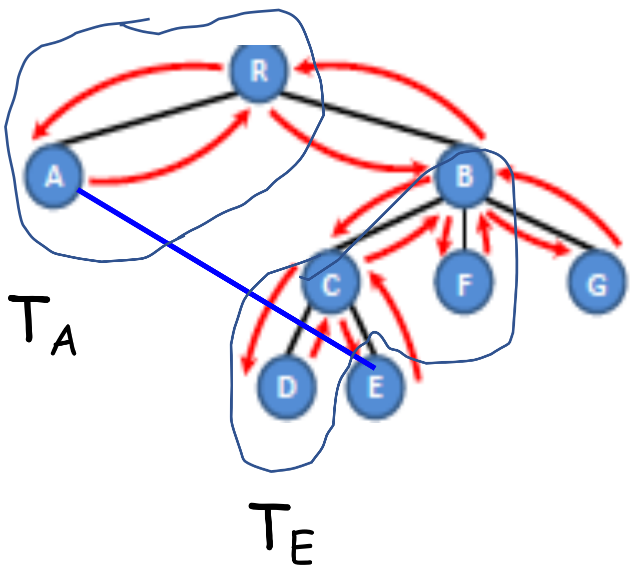
- ET trees *variant* to enable simpler path_edge
- Static parallel alg to compute
  - spanning forest (Gazit 1991)
  - spanning forest (E/F) relative to pre-existing forest

  returns minimal subset of edges in E which link trees in F)

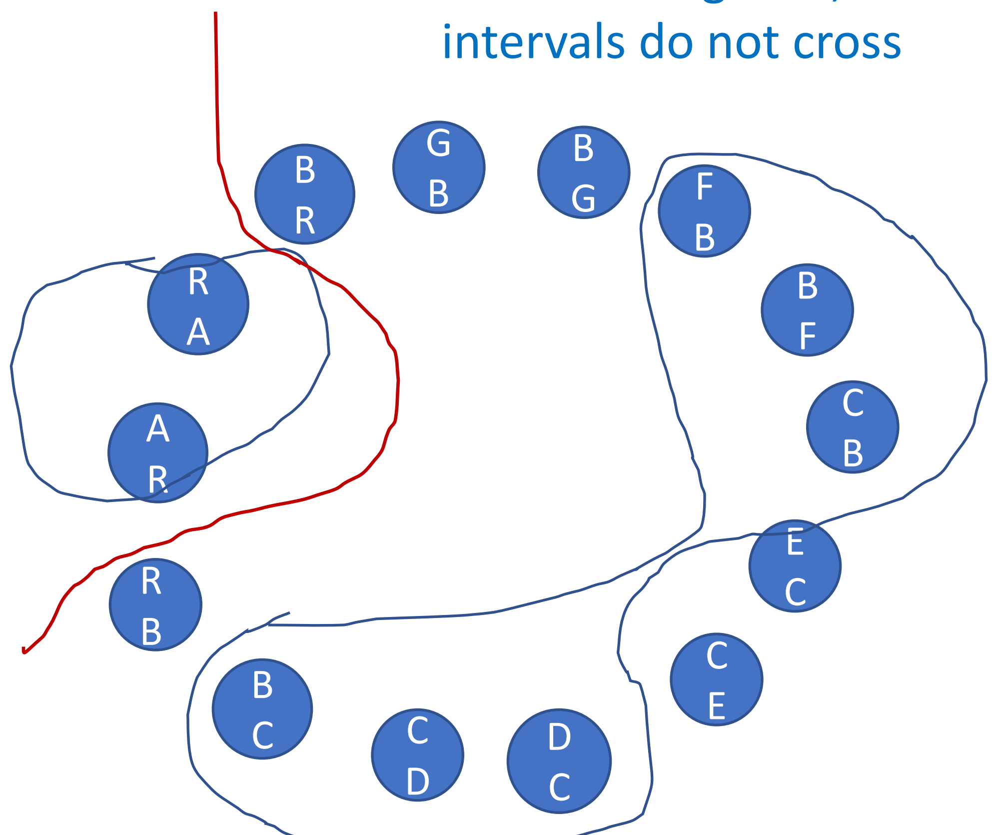
ET tree: path_edge (a,b)

Given T_a , T_b node disjoint trees in F_i and their corresponding intervals on the circle, path_edge(a,b)

returns edge {o, succ or pred(o)} where o is occurrence of node in T_a closest to T_b or vice versa.



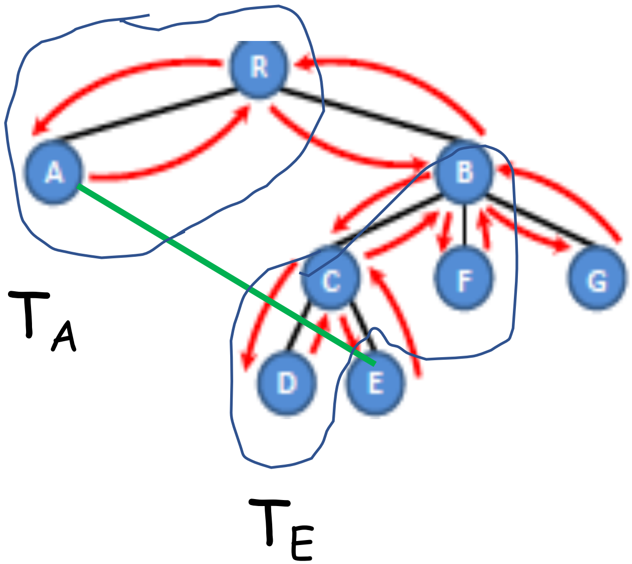
An interval for a subtree may not be contiguous, but the intervals do not cross

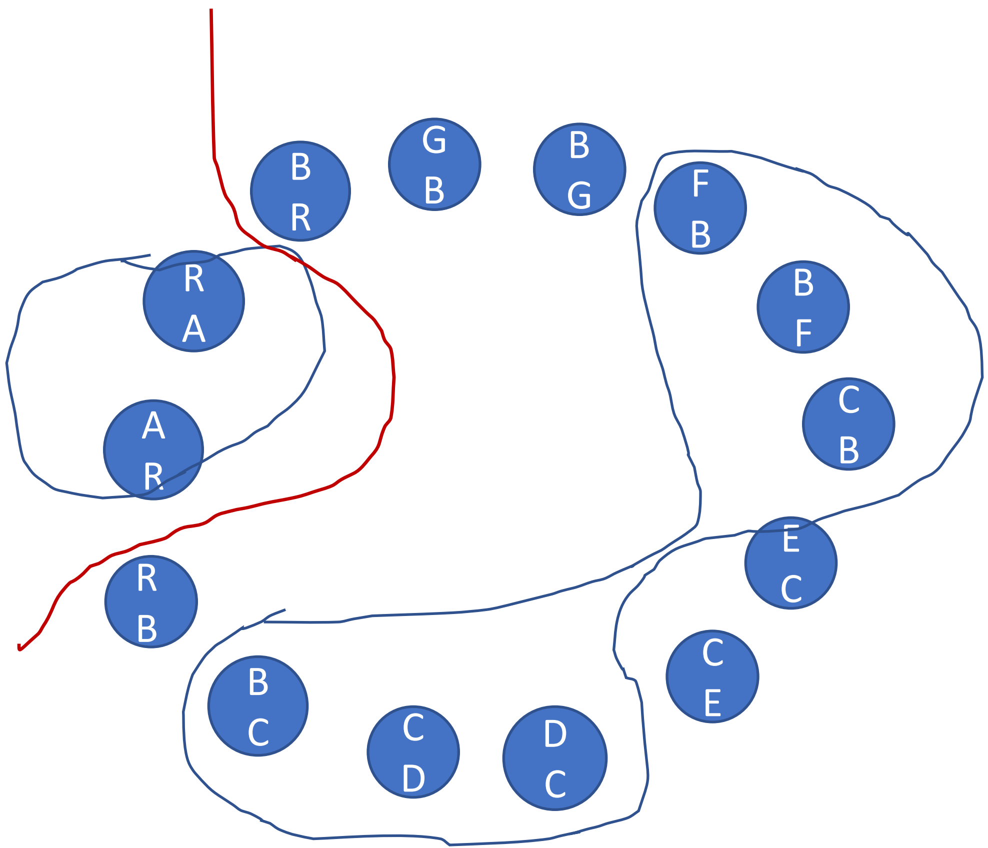


ET tree: path_edge (a,b)

Given T_a , T_b node disjoint trees in F_i and their corresponding intervals on the circle, path_edge(a,b)

returns edge {o, succ or pred(**o**)} where **o** is the occurrence of a node in T_a closest to T_b or vice versa.





Can binary search to find **o**

Batch parallel implementation of KKM

- Cutset data structure is easy to batch parallelize using batch parallel ET trees.
- O(log n) depth, $O(\log^2 n)$ work to maintain O(log n) word vectors which can be updated independently.

Insert also easy to batch parallelize as

cutset data structures c_0, $c_1 \ldots c_{top-1}$ can be updated independently using

batch parallel update values

Query uses batch parallel find trees

Batch parallel delete(S)

batchdelete (S) from all cutsets

for i=0 to top-1

$R_{j<I}$ = replacement edges found in lower tiers F_j , j<i

1. **Break cycles**: C<--batch_path_edge($R_{j<I}$,$F_{i+1} \setminus F_i$); make non-tree edges(C) in F_{i+1}
2. **make_tree_edge** ($R_{j<I}$) in F_{i+1}
3. **Reinsert subset of C to maintain connectivity**

   C'<-- Spanning Forest(C \F_{i+1}), make tree edges (C') in F_{i+1}

4. **Find new replacement edges**

   R<- Batch Find(T_v) for each v in V(S), T_v unmatched

   $R_{j<i+1}$ <--$R_{j<I}$ U R_i =Spanning_forest (R\F_{i+1})

   make_tree-edges(R_i) in F_{i+1}

Batch parallel delete(S)

batchdelete (S) from all cutsets

for i=0 to top-1

$R_{j<l}$ = replacement edges found in lower tiers F_j , j<i

1. **Break cycles**: C<--batch_path_edge($R_{j<l}$,$F_{i+1} \setminus F_i$); make non-tree edges(C) in F_{i+1}
2. make_tree_edge ($R_{j<l}$) in F_{i+1}
3. Reinsert subset of C to maintain connectivity

   C'<-- Spanning Forest(C \F_{i+1}), make tree edges (C') in F_{i+1}
4. Find new replacement edges

   R<- Batch Find(T_v) for each v in V(S), T_v unmatched

   $R_{j<i+1}$ <--$R_{j<l} \cup$ R_i =Spanning_forest (R\F_{i+1})

   make_tree-edges(R_i) in F_{i+1}

log n tree edges may change for each deleted edge->
k * log n* $\log^2$ n (log(1+n/k) work per deleted edge

Use of XOR method for asynchronous spanning tree construction with sublinear communication

Use of XOR method: Sublinear communication in distributed asynchronous network

Building a spanning tree in a distributed network,
where each node initially knows only its neighbors

Uses more properties of find_edge;

- randomness of edge selected
- approximation of cutset

XOR method

All nodes awake:

- Low degree ($< \sqrt{n} \log n$) nodes send to all their neighbors
- With prob $1/\sqrt{n}$) nodes choose themselves as special
- Specials send to all their neighbors

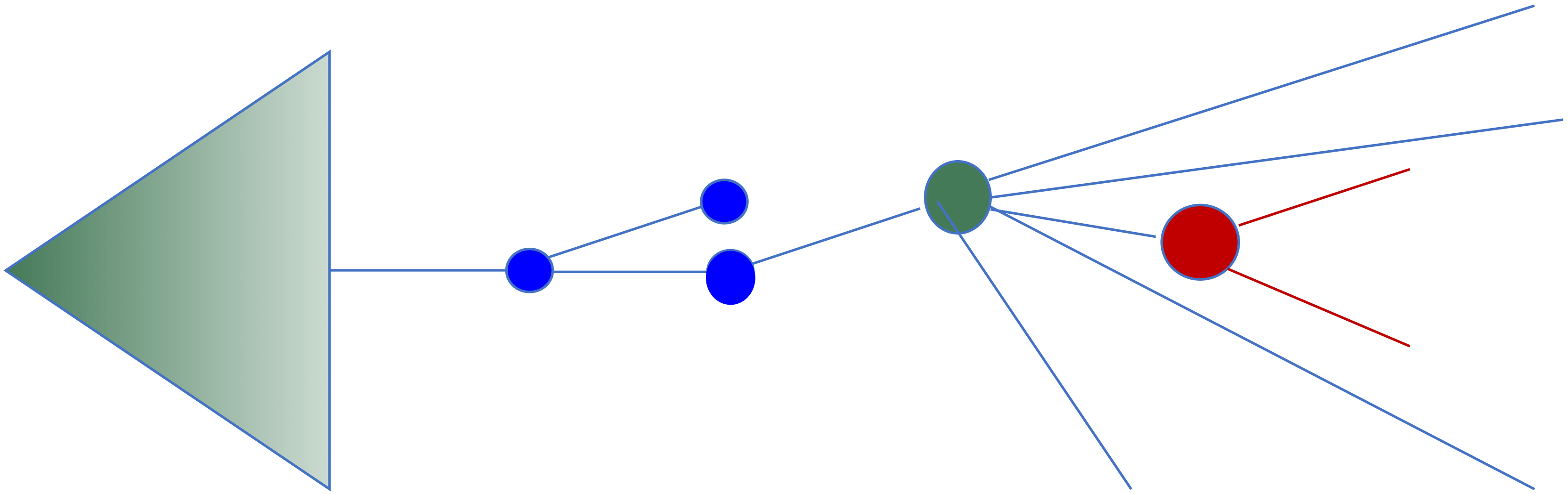
XOR method

Grow tree T from node 1 in phases:

Phase:

A. Expand T recursively:

- low degree nodes bring in their all neighbors
- high degree nodes wait to hear from at least one special (w.h.p) and
- Specials bring in all their neighbors.



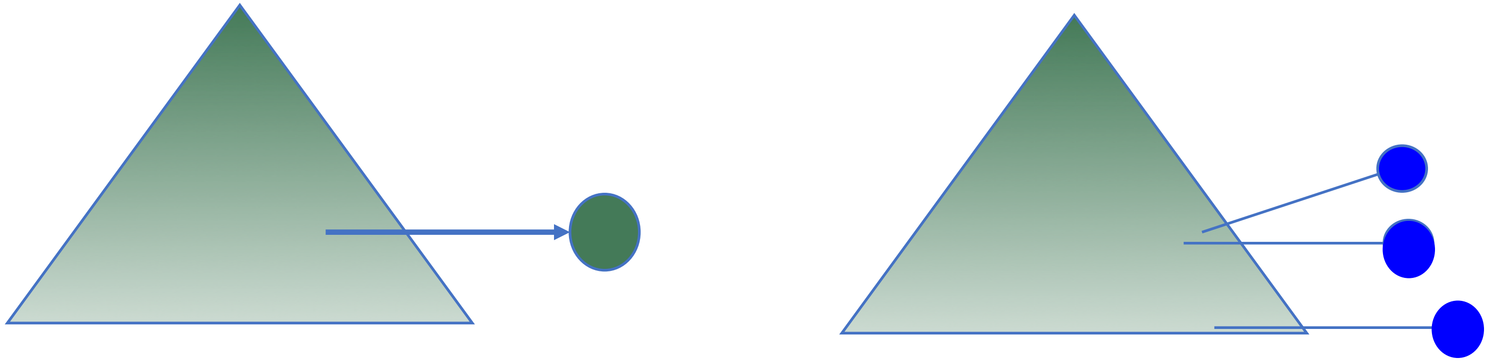
XOR method

Then use find_edge

Phase (cont'd):

B. Find an outgoing edge to a high degree node using find_tree

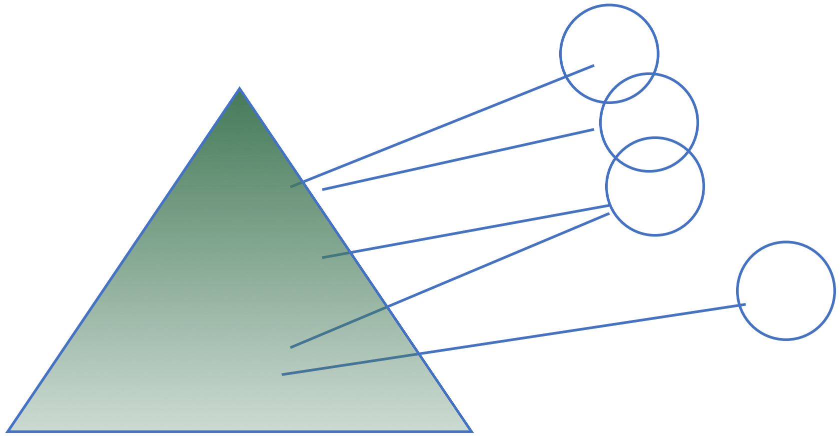
OR



C. WAIT until the low degree nodes contact T

How do you WAIT?? (Step C)

1. Do O(log n) find_edge's
   - if only edges to low degree nodes, then whp, edges to low degree nodes make up > ½ cutset; WAIT
   - if no edges found, end algorithm
   - else edge to high degree is found, end phase
2. Use find_edge to estimate cutset size K

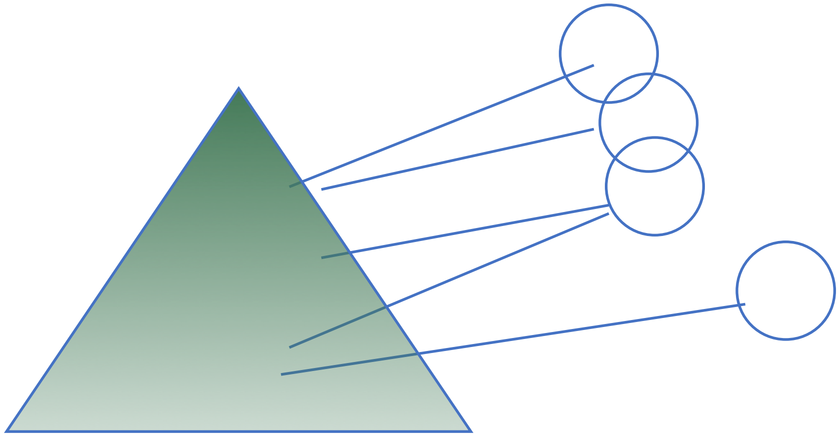


XOR method

Wait (Step C cont'd)

3. For each edge in cut found, trigger leader with prob c lg n/K

4. If leader receives c'lg n triggers, repeat Step C over undiscovered edges in cut (remove found edges from all samples)



Conclusion and open problems

- Improvement (polylog?) sequential worst case Las Vegas/Deterministic
- Improvement of sequential worst case Monte Carlo

  or allow for a nonoblivious adversary.

  $O(\log^3 n)$ for insertions

  $O(\log^4 n)$ for deletions
- In a distributed network, how much communication/time needed for constructing a tree of depth close to diameter of graph?

  $O(n^{3/2})$ for synchronous with depth $O(D+n^{1/2})$ (Ghaffari,Kuhn 2018; Gimr, Pandurangan

  $O(n^{3/2})$ for asynchronous with depth n, time $n^{3/2}$ (Mashreghi,K,2018)

  $m^{1+o(1)}$ communication in time $D^{1+o(1)}$ for asynchronous breadthfirst search tree (Awerbuch 1989)

Thank you!